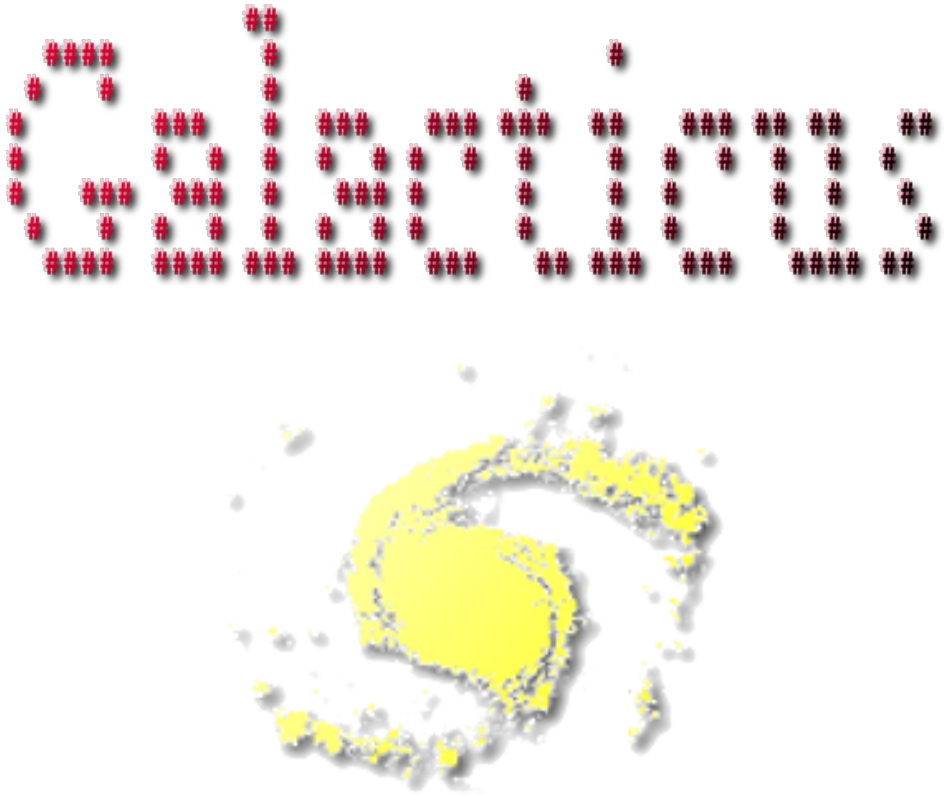

A semi-analytic galaxy formation code.

© 2009, 2010 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019 Andrew Benson

Contents

Part I.

Installation and Basic Use

1. About Galacticus

Galacticus is a semi-analytic model of galaxy formation. It solves equations describing how galaxies evolve in a merging hierarchy of dark matter halos in a cold dark matter universe. Galacticus has much in common with other semi-analytic models, such as the range of physical processes included and the type of quantities that it can predict.

In designing Galacticus our main goal was to make the code flexible, modular and easily extensible. Much greater priority was placed on making the code easy to use and modify than on making it fast. We believe that a modular and extensible nature is crucial as galaxy formation is an evolving science. In particular, key design features are:

Extensible methods for all functions: Essentially all functions within Galacticus are designed to be extensible following an Object Oriented methodology, meaning that you can write your own version and insert it into Galacticus easily. For example, suppose you want to use an improved functional form for the cold dark matter (CDM) halo mass function. You would simply write a new `haloMassFunction` class that computes this mass function, decorate it with a short directive (see §16.1) which explains to the build system how to insert connect this class into Galacticus. A recompile of the code will then incorporate your new function.

Extensible components for tree nodes: The basic structure in Galacticus is a merger tree, which consists of a linked tree of nodes which have various properties. Galacticus works by evolving the nodes forwards in time subject to a collection of differential equations and other rules. Each node can contains an arbitrary number of *components*. A component may be a dark matter halo, a galactic disk, a black hole etc. Each component may have an arbitrary number of *properties* (some of which may be evolving, others of which can be fixed). Galacticus makes it easy to add additional components. For example, suppose you wanted to add a "stellar halo" components (consisting of stars stripped from satellite galaxies). To do this, you would write a module which specifies the following for this component:

- Properties (their names, types, and ranks);
- Functions describing the differential equations which govern the evolution of the properties;
- Functions describing how the component responds to various events (e.g. the node becoming a satellite, a galaxy-galaxy merger, etc.);
- "Pipes" which allow for flows of mass/energy/etc. from one component to another.

Short directives embedded in this module explain to the Galacticus build system how to incorporate the new component. A recompile will then build your new component into Galacticus. Typically, a new component can be created quickly by copying an existing one and modifying it as necessary. Furthermore, multiple implementations of a component are allowed. For example, Galacticus contains a component which tracks the scale length of the dark matter halo. You could add a new component which additionally tracks the axis ratios of the (now triaxial) halo. A simple input parameter then allows you to select which implementation will be used in a given run.

Centralized ODE solver: Galacticus evolves nodes in merger trees by calling an ODE solver which integrates forwards in time to solve for the evolution of the properties of each component in a node. This means that you do not need to provide explicit solutions for ODEs (in many cases such

solutions are not available anyway) and timestepping is automatically handled to achieve a specified level of precision. The ODE solver allows for the evolution to be interrupted. A component may trigger an interrupt at any time and may do so for a number of reasons. A typical use is to actually create a component within a given node—for example when gas first begins to cool and inflow in a node the disk component must be created. Other uses include interrupting evolution when a merging event occurs.

1.0.1. Getting Galacticus

Downloads and installation instructions can be found at https://bitbucket.org/galacticusdev/galacticus/wiki/Home.

1.0.2. License

Copyright 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, Andrew Benson <abenson@carnegiescience.edu>

Galacticus is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

Galacticus is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with Galacticus. If not, see <http://www.gnu.org/licenses/>.

2. Running Galacticus

2.1. Configuration File

The file `galacticusConfig.xml`, is present, is used to configure Galacticus and provide useful information. It should have the following structure:

```
<config>
  <contact>
    <name>My Name</name>
    <email>me@ivory.towers.edu</email>
  </contact>
  <email>
    <host>
      <name>myComputerHostName</name>
      <method>smtp</method>
      <host>smtp-server.ivory.towers.edu</host>
      <user>myUserName</user>
      <passwordFrom>kdewallet</passwordFrom>
    </host>
    <host>
      <name>default</name>
      <method>sendmail</method>
    </host>
  </email>
</config>
```

The name and e-mail address in the `contact` section will be stored in any Galacticus models run—this helps track the provenance of the model. The `email` section determines how e-mail will be sent. Within this section, you can place one or more `host` elements, the `name` element of which specifies the host name of the computer to which these rules apply (the `default` host is used if no other match is found). For each host, the `method` element specifies how e-mail should be sent, either by `sendmail` or via `smtp`. For SMTP transport (which currently supports SSL connections only), you must specify the `host` SMTP server, `user` name. The `passwordFrom` element specifies how the password for the SMTP log in should be obtained. If set to `input` then the user will be prompted for the password as needed. Alternatively, if you use the KDE desktop and the KDEWallet password manager, setting `passwordFrom` to `kdewallet` will cause the password to be stored in the KDE wallet and retrieved from there subsequently.

2.2. Parameter Files

Galacticus requires a file of parameters to be given as a command line argument. The parameter file is an XML file (which makes it easy to manipulate and construct these files from within many languages, e.g. Perl) with the following structure:

```
 <parameters>
```

```
  <version>0.9.4</version>
  <formatVersion>2</formatVersion>
  <parameter1Name value= "parameter1Value" />
  <parameter2Name>
    <value>parameter2Value</value>
  </parameter2Name>
  <parameter3Name value= "parameter3Value" >
     <subParameter1Name value= "subParameter1Value" />
     <subParameter2Name value= "subParameter2Value" />
     .
     .
     .
  </parameter3Name>
  .
  .
  .
  <parameter4Name value= "parameter4Value" id="myRefParam" >
     <subParameter1Name value= "subParameter1Value" />
     <subParameter2Name value= "subParameter2Value" />
     .
     .
     .
  </parameter3Name>
  .
  .
  .
  <parameter4Name idRef="myRefParam"/>
  .
  .
  .
</parameters>
```

Each named element must have a `value` attribute (preferred), or else contains a value element, which contains desired value respectively. The value can be a number, word(s) or an array of space-separated numbers or words. Parameters are used to control the values of numerical parameters and also to select methods and other options. If a parameter is not specified in the file a default value (hard coded into Galacticus) will be used instead. The default values have been chosen to produce a realistic model of galaxy formation, but may change as Galacticus evolves. Parameters may have sub-parameters embedded within them, as in the example above.

Sub-parameters are used in object composition within Galacticus. For example, the following would specify that linear growth of cosmological large scale structure should be modeled using the `simple` method:

```
<linearGrowthMethod value="simple">
  <cosmologyParametersMethod value="simple">
    <HubbleConstant   value="70.0" />
    <OmegaMatter      value="0.31"  />
    <OmegaDarkEnergy  value="0.69"  />
    <OmegaBaryon      value="0.045"/>
    <temperatureCMB   value="2.725"/>
  </cosmologyParametersMethod>
```

```
    <cosmologyFunctionsMethod value="matterLambda"/>
  </linearGrowthMethod>
```

The linear growth function object requires knowledge about the cosmological parameters and model. In the above, we specify this explicitly by including a definition of the cosmological parameter object and cosmological functions object that our linear growth function object should use. Note that the cosmological functions object also requires knowledge of the cosmological parameters. When the `cosmologyFunctionsMethod` object is built from the above definition it will first check for a cosmological parameters object defined in its own subparameters. Since it does not find one in this instance it will check for a cosmological parameters definition in its parent object (the `linearGrowthMethod` element) and, in this case, will use that definition. If no definition were to be found in any parent element, a default set of cosmological parameters would be used instead[1].

Parameters can also be defined with an "`id`" attribute. Such parameters are targets for pointers elsewhere in the file, and as such are inactive (i.e. they will be ignored by Galacticus except as targets for pointers). A pointer to a target is created by specifying an element with the same parameter name and an "`idRef`" element with a value equal to that of the `id` element of the target. The pointer then acts as a regular parameter, adopting the value and subparameters of the target. Targets can be targetted by multiple pointers. This allows for a single object to be shared between multiple other objects.

The optional `version` element specifies which version of Galacticus this parameter file is intended for. The optional `formatVersion` element specifies the parameter file version number (the current standard for parameter files is version 2). While optional, these elements can be useful when migrating parameter files between versions of Galacticus (see §6.5).

All parameter values (both those specified in this file and those set to default) used during a Galacticus run are output to the `Parameters` group within the Galacticus output file. The script `scripts/aux/Extract_Parameter_File.pl` will, if given a Galacticus output file, extract the parameters from it and output them into an XML file suitable for re-input into Galacticus. If parameters are present in the parameter file which do not match any known parameter in Galacticus then a warning message, listing all unknown parameters, will be given when Galacticus is run. Note that this will *not* prevent Galacticus from running—sometimes it is convenient to include parameters which are not used by Galacticus, but which might be used by some other code.

2.2.1. Validating Parameter Files

A script, `scripts/aux/validateParameters.pl`, is provided to validate parameter files and thereby ensure that they are consistent with Galacticus's expectations and requirements. To use simply execute:

```
scripts/aux/validateParameters.pl myParameters.xml
```

No output (and an exit value of 0) indicates a valid parameter file. Invalid parameter files will result in an exit value other than 0 and will produce error messages that should help to track down the problem with the file.

2.2.2. Generating Parameter Files

Some scripts are provided which assist in the generation of parameter files. These are located in the `scripts/parameters/` folder and are detailed below:

[1] This approach allows a direct connection to be made between the structure of the input parameter XML file and the internal object hierarchy used by Galacticus, allowing very fine-grained control over the composition of Galacticus functionality. In particular it permits easy construction of objects which work by modifying results from other objects, such as the `schneider2015` model for dark matter halo concentrations (§12.11.3).

`cosmologicalParametersMonteCarlo.pl` This script will generate a set of cosmological parameters drawn at random from the WMAP-9 constraints Hinshaw et al. [2012]. It uses the covariance matrix (currently defined in `data/Cosmological_Parameters_WMAP-9.xml`) to produce correlated random variables[2]. The generated parameters are printed to standard output as Galacticus-compatible XML.

2.3. Running Galacticus

Galacticus is running using

```
Galacticus.exe [<parameterFile>]
```

where `parameterFile` is the name of the file containing a list of parameter values for Galacticus. Galacticus will display messages indicating its progress as it runs (the verbosity can be controlled with the `verbosityLevel` parameter). Usually, the Galacticus executable should be invoked from the directory in which it was built. However, you can choose to set the environment variable `GALACTICUS_ROOT_V091` to the full path to the build directory, in which case the Galacticus executable can be invoked from anywhere and will access all required files and scripts relative to this path. This can allow multiple users to all make use of the same Galacticus install.

2.3.1. Writing Data To a Temporary File

When running Galacticus on a compute cluster it is often advantageous to have output written to a local scratch disk during run time and only moved to networked storage after the run is complete. (Otherwise, Galacticus will perform many small writes to networked storage which can result in extremely slow run times.) To do this, simply set the parameter `[galacticusOutputScratchFileName]` to the full path of a file to write to on local scratch space. During the run, data will be written to this file. After the run is finished, Galacticus will move this file to its permanent location as specified by the parameter `[galacticusOutputFileName]`.

2.3.2. Restarting A Crashed Run

If Galacticus crashes, it can be useful to restart the calculation from just prior to the crash to speed the debugging process. Galacticus has functionality to store and retrieve the internal state of any modules and to recover this to permit such restarting. Currently, this is implemented with the `build` and `read` methods of merger tree construction, such that the internal state is stored prior to commencing the building or reading of each tree, thereby allowing a calculation to be restarted with the tree that crashed. More general store/retrieve behavior is planned for future releases.

To cause Galacticus to periodically store its internal state include the following input parameter:

```
<stateFileRoot value="galacticusState" />
```

This will cause the internal state to be stored to files `galacticusState.state` and `galacticusState.fgsl.state` prior to commencing building each merger tree. Should a tree crash then replace this input parameter with:

```
<stateRetrieveFileRoot           value="galacticusState" />
<mergerTreeBuildTreesBeginAtTree value="N"               />
```

[2]Note that this does not capture the full details of the correlations between parameters, since it uses just the covariance matrix. For a more accurate calculation the full Monte Carlo Markov Chains used in the WMAP-9 parameter fitting should be used instead.

where `N` is the number of the tree that crashed. This will cause calculations to begin with tree `N` and for the internal state to be recovered from the above mentioned files. The resulting tree and all galaxy formation calculations should therefore proceed just as in the original run (and so create the same crash condition).

OpenMP

When running a model in parallel using OpenMP, a separate state file will be written for each thread, with the thread number appended to the end of each state file name. For debugging purposes, it is suggested that a crashed OpenMP run be restarted using just a single thread. To do this, change the appended thread number on the state files corresponding to the thread which crashed to 0 such that they will be used by the single thread when the run is restarted.

2.3.3. Running Grids of Models

You can easily write your own scripts to generate parameter files and run Galacticus on these files. An example of such a script is `scripts/aux/launch.pl`. This script will loop over a sequence of parameter values, generate appropriate parameter files, run Galacticus using those parameters and analyze the results. This script currently supports running of Galacticus on a local machine, via a PBS queue (as multiple jobs or a single job), or on a Condor cluster. To run the script simply enter:

```
./scripts/aux/launch.pl <runFile>
```

This will launch a single instance of the script. Multiple instances can be launched and will share the work load (i.e. they will not attempt to run a model which another instance is already running or has finished). If multiple instances are to be launched on multiple machines a command line option to `launch.pl` can be used to ensure that they do not duplicate work. Adding `-instance 2:4` for example will tell the script to run only the second model from each block of four models it finds. Launching for `launch.pl` scripts on four different machines with `-instance 1:4`, `-instance 2:4`, `-instance 3:4` and `-instance 4:4` will then divide the models between those machines.

The `runFile` is an XML file with the following structure:

```
<parameterGrid>
 <modelRootDirectory>models.new</modelRootDirectory>
 <baseParameters>newBestParametersQuick.xml</baseParameters>
 <compressModels>no</compressModels>
 <splitModels>4</splitModels>

 <launchMethod>pbs</launchMethod>

  <local>
   <threadCount>3</threadCount>
   <ompThreads>4</ompThreads>
  </local>

 <condor>
  <galacticusDirectory>/home/condor/Galacticus/v0.9.3</galacticusDirectory>
  <universe>vanilla</universe>
  <environment>LD_LIBRARY_PATH=/usr/lib:/usr/lib64:/usr/local/lib</environment>
  <requirement>Memory &gt;= 1000 &amp;&amp; Memory &lt; 2000</requirement>
  <transferFile>{PWD}/myFile.data</transferFile>
```

```
 <wholeMachine>true</wholeMachine>
 <postSubmitSleepDuration>5</postSubmitSleepDuration>
 <jobWaitSleepDuration>10</jobWaitSleepDuration>
</condor>

<pbs>
 <scratchPath>/scratch/me</scratchPath>
 <wallTime>48:00:00</wallTime>
 <memory>3gb</memory>
 <ompThreads>8</ompThreads>
 <queue>standard</queue>
 <maxJobsInQueue>10</maxJobsInQueue>
 <mpiLaunch>yes</mpiLaunch>
 <mpiRun>/opt/openmpi/bin/mpirun</mpiRun>
 <environment>LD_LIBRARY_PATH=/home/me/software/Galacticus/Tools/lib64:$LD_LIBRARY_PATH</environment>
 <postSubmitSleepDuration>10</postSubmitSleepDuration>
 <jobWaitSleepDuration>60</jobWaitSleepDuration>
</pbs>

<monolithicPBS>
 <mpiLaunch>yes</mpiLaunch>
 <nodes>1</nodes>
 <threadsPerNode>12</threadsPerNode>
 <ompThreads>6</ompThreads>
 <jobWaitSleepDuration>60</jobWaitSleepDuration>
 <analyze>no</analyze>
 <environment>LD_LIBRARY_PATH=/home/me/software/Galacticus/Tools/lib64:$LD_LIBRARY_PATH</environment>
 <includePath>/my/include/path</includePath>
 <libraryPath>/opt/sgi/mpt/mpt-2.04/lib</libraryPath>
 <shell>csh</shell>
 <pbsCommand>source /usr/share/modules/init/csh</pbsCommand>
 <pbsCommand>module load mpi-sgi/2.04_64</pbsCommand>
</monolithicPBS>

<parameters>
 <label>modelLabel</label>
 <stabilityThresholdStellar value="1.1"/>
 <stabilityThresholdStellar value="0.9"/>
</parameters>

<parameters>
 <starFormationFeedbackDisksMethod value="powerLaw">
  <exponent value="2.5"/>
  <exponent value="3.0"/>
 </starFormationFeedbackDisksMethod>
 <starFormationFeedbackDisksMethod value="creasey2012"/>
</parameters>

<parameters>
```

```
  <imfSelectionMethod value="fixed">
    <imfSelectionFixed value="Chabrier" parameterLevel="top"/>
    <imfSelectionFixed value="Salpeter" parameterLevel="top"/>
  </imfSelectionMethod>
  <imfSelectionMethod value="diskSpheroid">
    <imfSelectionDisk     value="Chabrier"  parameterLevel="top"/>
    <imfSelectionSpheroid value="Kennicutt" parameterLevel="top"/>
  </imfSelectionMethod>
 </parameters>

 <parameters>
   <coolingFunctionMethod value="summation">
     <coolingFunctionMethod value="atomicCIECloudy"             iterable="no"/>
     <coolingFunctionMethod value="CMBCompton"                  iterable="no"/>
     <coolingFunctionMethod value="molecularHydrogenGalliPalla" iterable="no"/>
   </coolingFunctionMethod>
 </parameters>

</parameterGrid>
```

Each `parameters` block contains a list of parameters following the format used in standard Galacticus parameter files, with the difference that each parameter can appear multiple times, each time with a different `value` attribute, as is the case for `stabilityThresholdStellar` in the first `parameters` element in the above. A model will be run for all possible combinations of these values. For nested parameters with multiple values, all possible values of these parameters will be looped over when, and only when, the appropriate value of the containing parameter is being used. For example, in the second `parameters` element in the above example, models will be run with subparameter `[exponent]=2.5` and `3.5` for the `starFormationFeedbackDisksMethod` element only when `[starFormationFeedbackDisksMethod]=powerLaw` and not when `[starFormationFeedbackDisksMethod]=creasey2012`. It is also possible to specify that subparameter should be promoted to the top-level of the parameter file. In the third `parameters` element in the above example, `imfSelectionFixed` will take on values of `Chabrier` and `Salpeter` only when `imfSelectionMethod=fixed`, and the `imfSelectionFixed` element will be promoted from a subparameter of `imfSelectionMethod` to the top-level of the parameter file due to the presence of the `parameterLevel="top"` attribute. Finally in some cases a parameter which appears multiple times is not to be iterated over. In the fourth `parameters` element in the above example, this is the case for the `coolingFunctionMethod` subparameters. The addition of an `iterable="no"` attribute specifies that these parameters are not to be iterated over, but simply left as they are.

Some variables, which are expanded at run time, are available. These include:

`%%galacticusOutputPath%%` This will be expanded to the output path of a model. Useful for specifying paths for any additional output.

By default, each model is output into a sequentially numbered directory within the `./models` directory. By default, these directories have the prefix `galacticus`. This can be changed by including a `label` element inside a `parameters` block, in which case the content of the `label` element will be used as the prefix. This root directory can be modfified by the optional `modelRootDirectory` element. Additionally, a set of base parameters can be read from a file specified by the `baseParameters` file—these will be read before each model is run and before any variations in parameters for the specific model are applied. As such, it defines the default model around which parameter variations occur. Additional options that may be present in the file (as elements within the `parameterGrid` element) are:

`doAnalysis` If set to "no" then no analysis scripts will be run on completed models, otherwise, they will be. Optionally, the analysis script to run can be specified via the `analysisScript` element (see §2.3.4);

`emailReport` If set to "yes" a report will be e-mailed to the address specified in `galacticusOptions.xml` when a model fails. Otherwise, the report will be written to standard output instead.

`compressModels` If "no" then models are not compressed after being run. Otherwise, the contents of the model output directory will be compressed using `bzip2`.

`splitModels` If set to an integer larger than 1, each Galacticus model will be split into that number of jobs, and those jobs will be launched (using the selected method) independently. Once finished, the outputs from these split models will be merged back into a single model. This allows, for example, effectively distributing a single Galacticus model over multiple nodes of a PBS cluster.

The method by which to launch jobs must be specified in the `launchMethod` element. Currently available options are:

`local` The models will be run on the local machine. Two additional options can be specified within a `local` XML block:

- `threadCount` The number of individual model threads to be launched.
- `ompThreads` The number of OpenMP threads to be used by each model.

`pbs` Jobs will be submitted to a `PBS` batch queue system. The following options are available and can be specified within a `pbs` XML block:

- `scratchPath` An optional path to which the model output will be written at run time. At the completion of each run, the data will be transferred to the usual output location. This is useful to avoid network I/O during run time;
- `wallTime` A limit on the wall time allowed for each model (optional);
- `memory` A limit on the memory allowed for each model (optional);
- `ompThreads` The number of OpenMP threads to use for each model (optional). This is used to request an appropriate number of processors per node;
- `queue` The name of the queue to submit the jobs to (optional);
- `maxJobsInQueue` The maximum number of jobs to place in the queue. Additional jobs will be held and submitted once the number of jobs in the queue drops below this value (optional);
- `mpiLaunch` If set to "`yes`" then the `mpirun` command will be used to launch a single copy of Galacticus (which may then spawn multiple OpenMP threads). If instead set to "`no`" then Galacticus is launch without the use of the `mpirun` command. Some systems will limit a code launched with `mpirun` to using just a single CPU (even if multiple OpenMP threads are spawned). In such cases, setting this option to "`no`" should permit multiple CPUs to be utilized.
- `mpiRun` The path to the `mpirun` executable (optional—if not present, `mpirun` must be in `PATH`);
- `environment` Any settings here are set in each PBS job in order to set appropriate environment variables on the machine where a job is executed;
- `analyze` If set to "`yes`" then analysis (if any) will be performed as part of the PBS job. Otherwise, analysis is performed by the submitting machine.
- `postSubmitSleepDuration` The time (in seconds) to wait after submitting each job. This prevents flooding the PBS queue manager with a large number of jobs in rapid succession.

`jobWaitSleepDuration` The time (in seconds) to sleep between successive checks of the PBS queue to see if any of the submitted jobs have finished.

`monolithicPBS` A single job will be submitted to a `PBS` batch queue system. This job will internally run multiple copies of Galacticus each with a different set of parameters. The following options are available and can be specified within a `monolithicPBS` XML block:

`nodes` The total number of nodes to use for the PBS job.

`threadsPerNode` The number of threads per node to use for the PBS job.

`ompThreads` The number of OpenMP threads to use for each model (optional). This is used to request an appropriate number of processors per node, and must be an factor of `threadsPerNode`;

`scratchPath` An optional path to which the model output will be written at run time. At the completion of each run, the data will be transferred to the usual output location. This is useful to avoid network I/O during run time;

`wallTime` A limit on the wall time allowed for each model (optional);

`memory` A limit on the memory allowed for each model (optional);

`queue` The name of the queue to submit the jobs to (optional);

`mpiRun` The path to the `mpirun` executable (optional—if not present, `mpirun` must be in `PATH`);

`environment` Any settings here are set in each PBS job in order to set appropriate environment variables on the machine where a job is executed;

`analyze` If set to "`yes`" then analysis (if any) will be performed as part of the PBS job. Otherwise, analysis is performed by the submitting machine.

`postSubmitSleepDuration` The time (in seconds) to wait after submitting each job. This prevents flooding the PBS queue manager with a large number of jobs in rapid succession.

`jobWaitSleepDuration` The time (in seconds) to sleep between successive checks of the PBS queue to see if any of the submitted jobs have finished.

`condor` Jobs will be submitted to a `Condor` cluster. The following options are available and can be specified within a `condor` XML block:

`galacticusDirectory` When a Galacticus job is submitted to a Condor cluster the Galacticus executable and the input parameter file are transferred to the machine where the job runs. Other files, such as data files, are not transferred. Therefore, they must be already present on any remote machine on which the job can run. This option specifies where a complete Galacticus installation can be found on the remote machine. If not present, it defaults to `/home/condor/Galacticus/v0.9.0`;

`universe` Specifies to which Condor universe jobs should be submitted. Allowed options are "vanilla" and "standard". If the standard universe is to be used then Galacticus must have been linked with `condor_compile`—the `Makefile` allows this if the relevant lines are uncommented;

`environment` Any settings here are passed to Condor's `environment` option in order to set appropriate environment variables on the machine where a job is executed;

`requirement` Any setting here is passed to Condor's `requirements` option to specify requirements for each job. Multiple `requirement` entries will be combined (using logical and).

`transferFile` Any files listed here will be transferred the Condor worker (and so will be accessible from the path in which Galacticus is running). The macro `{PWD}` will be automatically expanded to the present working directory. Multiple `transferFile` entries can be given.

`wholeFile` Setting this option to `true` will add `+RequiresWholeMachine = True` to the Condor submit file. If Condor has been configured to allow jobs to take over a whole machine[3], this will cause jobs to do so. This is useful if you want to run OpenMP Galacticus on a Condor cluster.

`postSubmitSleepDuration` The time (in seconds) to wait after submitting each job. This prevents flooding the Condor queue manager with a large number of jobs in rapid succession.

`jobWaitSleepDuration` The time (in seconds) to sleep between successive checks of the Condor queue to see if any of the submitted jobs have finished.

In addition to the `galacticus.hdf5` output file, each model directory will contain a file `newParameters.xml` which contains the parameters used to run the model and `galacticus.log` which contains any output from Galacticus during the run.

If present, the file `galacticusConfig.xml`, described in §2.1, is parsed for configuration options. If the `contact` element is present, the listed name and e-mail address will be used to determine who should receive error reports should a model crash. The error report will contain the host name of the computer running the model, the location of the model output and the log file (which may be incomplete if output is being buffered). Additionally, any core file produced will be stored in the model directory for later perusal, and the state files (see §2.3.2) for the run can also be found in the model directory.

2.3.4. Analysis of Models

The `Run_Galacticus.pl` script will automatically run `scripts/analysis/Galacticus_Compute_Fit.pl` on each model to generate plots and fitting data unless `doAnalysis=no` is set in the `runFile` (see §2.3.3). This script, which can also be running manually using

```
./scripts/analysis/Galacticus_Compute_Fit.pl <galacticusFile> <outputDirectory> [<analysisFile>]
```

where `galacticusFile` is the name of the Galacticus output file to analyze and `outputDirectory` is the directory into which plots and fitting data should be placed, reads the file `<analysisFile>` (or `data/Galacticus_Compute_Fit_Analyses.xml` if `<analysisFile>` is not specified) which has the following structure:

```
<analyses>
 <analysis>
   <script>scripts/plotting/Plot_HI_Mass_Function.pl</script>
   <weight>1.0</weight>
 </analysis>
 <analysis>
   <script>scripts/plotting/Plot_K_Luminosity_Function.pl</script>
   <weight>1.0</weight>
 </analysis>
 .
 .
 .
</analyses>
```

Each `analysis` element contains the name of a script to run to perform some analysis and a weight to be given to the results of this analysis when combining results to get a net goodness of fit. Each script listed will be run and is expected to have accept arguments of the form:

[3] As described here for example.

```
My_Analysis_Script.pl <galacticusFile> <outputDirectory> <showFit>
```

where the `showFit` argument can be 0 or 1 and, if set to 1, the script should output an XML chunk to standard output giving details of its fitting analysis. This chunk should have the form:

```
<galacticusFit>
  <name>Description of this analysis</name>
  <chiSquared>24.5</chiSquared>
  <degreesOfFreedom>19</degreesOfFreedom>
  <fileName>Output_File_Name.pdf</fileName>
</galacticusFit>
```

where `chiSquared` and `degreesOfFreedom` are the fitting results. All such data returned from fitting scripts will be collated by `Galacticus_Compute_Fit.pl`, augmented with the weight value and the net goodness of fit determined. All of this information is then output to `galacticusFits.xml` in the selected output directory.

Performing Other Analysis

If `<doAnalysis >=yes` and `<analysisFile>` is set to something other than an XML file it is assumed that this is an analysis script that should be run directly. The script will be executed with the output directory for the Galacticus model as the first and only argument.

2.3.5. Processing Individual Merger Trees In Parallel

By default, Galacticus utilizes the available parallel threads to process multiple merger trees simultaneously, with one tree processed by each thread. When the total number of trees to be processed is large, and there are not a small number of outlier trees with masses very much larger than the other trees, this approach generally results in good parallel efficiency.

However, in cases where a small number of trees are much more massive than any other (or are just slow to process for some other reason) it may be more efficient to have multiple parallel threads process each tree. To achieve this, set `[treeEvolveSingleForest]=true`. In this case, trees are processed sequentially, with multiple threads assigned to each tree. To do this, a tree is broken up into a set of time slices, or "sections". The number of sections between each successive output (or between the earliest node in the tree and the first output) is specified by the `[treeEvolveSingleForestSections]` parameter. Individual branches of the tree within each section are assigned to parallel threads. *Note that this results in valid evolution only if the evolution of disjoint tree branches are independent of each other.*

2.3.6. Running Models in "Embarrassingly Parallel" Mode

While Galacticus is parallelized via OpenMP it is also possible to split a given model across several "worker" CPUs on one or more computers. The trees to be processed will be shared between these workers and the results can be later recombined. To use this "poor man's" parallelization, add the following to a model parameter file:

```
  <treeEvolveWorkerCount  value="N" />
  <treeEvolveWorkerNumber value="i" />
```

where `N` is the total number of workers to be used and `i` is the number of this worker (ranging from 1 to `N`). You can generate these individual input parameter files from a single base parameter file using:

```
scripts/aux/Split_Models_For_Workers.pl <parameterFile> <workerCount>
```

where `<parameterFile>` is the name of the base parameter file and `<workerCount>` is the number of workers required. The script will create an input file for each worker (input files will have the same name as the base parameter file but with a "`_N`", where `N` is the worker number, inserted before the "`.xml`"). Output file name for each worker will be the same as specified in the base parameter file, but with a "`_N`", where `N` is the worker number, inserted before the "`.hdf5`".

Once all workers have finished, their outputs can (if required) be combined into a single output file using the `Merge_Models.pl` script as follows:

```
./scripts/aux/Merge_Models.pl <model1> <model2> .... <modelOutput>
```

where `model1` etc. are the names of the various output files and `modelOutput` is the file into which the combined results should be placed. The `Merge_Models.pl` script will combine all merger trees into the output file and will additionally cumulate any data in the `globalHistory` groups in these files. The UUIDs of the merged files (see §4.1.1) will be concatenated (with a ":" separator) and placed into the `UUIDs` attribute of the new file. Additionally, a new UUID will be generated and stored in the `UUID` attribute of the new file.

2.3.7. Limiting the Load Average

If `[treeEvolveLimitLoadAverage]=true` then Galacticus will attempt to keep the load average of the system under `[treeEvolveLoadAverageMaximum]` by waiting to run trees if the current load average exceeds this value. `[treeEvolveLoadAverageMaximum]` can be set to the numerical maximum load average desired or, alternatively, can be set to `processorCount` in which case the number of processor cores present on the system will be used for `[treeEvolveLoadAverageMaximum]`.

Thread Locking via Semaphores

An alternative is to use Linux's semaphoring system to share processors between multiple instances of Galacticus running on a single machine. To activiate this mode, set `[treeEvolveThreadLock]` to the total number of threads that should be active at any one time across all instances of Galacticus running on a machine. You can also set this parameter to `processorCount` to set a a value equal to the total number of processors on the machine. With this parameter set, each instance of Galacticus running on a machine will request a semaphore for each thread that it attempts to run, with the maximum number of semaphores set to the value of `[treeEvolveThreadLock]`. This means that, if one instance of Galacticus has claimed 6 semaphores when `[treeEvolveThreadLock]=8`, then a second instance of Galacticus will only run 2 threads. Once the first instance releases its semaphores, the second instance will claim them and begin running more threads.

2.4. Tasks

By default, Galacticus executes a single "task" when run—to evolve a set of merger tree forests and output the resulting halo and galaxy properties. Different tasks can be performed however, controlled by the `taskMethod` parameter.

2.4.1. `evolveForests`

This is the standard task performed by Galacticus. It takes a set of merger tree forests, evolves them, and outputs the resulting halo and galaxy properties.

2.4.2. multi

This task allows multiple sub-tasks to be run. For example:

```
<taskMethod value="multi">
  <taskMethod value="haloMassFunction"/>
  <taskMethod value="evolveForests"   />
</taskMethod>
```

would result in the `haloMassFunction` task being performed, followed by the `evolveForests` task.

2.4.3. haloMassFunction

The `haloMassFunction` task will tabulate the halo mass function and related quantities and output them to the main Galacticus output file. The tabulation is controlled by three sub- parameters:

`[haloMassMinimum]` The lowest mass halo (in units of $M_\odot$) at which to tabulate;

`[haloMassMaximum]` The highest mass halo (in units of $M_\odot$) at which to tabulate;

`[pointsPerDecade]` The number of points per decade of halo mass at which to tabulate.

T The halo mass function is computed at each output specified by the `[outputRedshifts]` parameter, and written to the corresponding `Outputs/OutputN` group with the following structure:

```
+-> Outputs
|   |
|   +-> OutputN
|       |
|       +-> massHaloCharacteristic         [attribute]
|       |
|       +-> criticalOverdensity            [attribute]
|       |
|       +-> outputExpansionFactor          [attribute]
|       |
|       +-> growthFactor                   [attribute]
|       |
|       +-> outputRedshift                 [attribute]
|       |
|       +-> outputTime                     [attribute]
|       |
|       +-> virialDensityContrast          [attribute]
|       |
|       +-> haloMass                       [dataset]
|       |
|       +-> haloSigma                      [dataset]
|       |
|       +-> haloAlpha                      [dataset]
|       |
|       +-> haloPeakHeightNu               [dataset]
|       |
|       +-> haloMassFunctionM              [dataset]
|       |
```

```
|       +-> haloMassFunctionLnM            [dataset]
|       |
|       +-> haloMassFunctionLnMBinAveraged [dataset]
|       |
|       +-> haloMassFunctionNuFNu          [dataset]
|       |
|       +-> haloMassFunctionCumulative     [dataset]
|       |
|       +-> haloMassFractionCumulative     [dataset]
|       |
|       +-> subhaloMassFunctionCumulative  [dataset]
|       |
|       +-> haloBias                       [dataset]
|       |
|       +-> haloVirialRadius               [dataset]
|       |
|       +-> haloVirialTemperature          [dataset]
|       |
|       +-> haloVirialVelocity             [dataset]
|       |
|       +-> haloScaleRadius                [dataset]
|       |
|       +-> haloVelocityMaximum            [dataset]
|
+-> cosmology
|   |
|   +-> densityCritical                    [attribute]
|
+-> powerSpectrum
    |
    +-> massHalfMode                       [attribute]
```

The `Outputs/OutputN` groups contain atrributes and datasets which give properties at the corresponding output time as follows:

`massHaloCharacteristic` The characteristic mass scale (in units of $\mathrm{M}_\odot$), M_*, at which $\sigma(M) = \delta_\mathrm{c}(z)$;

`criticalOverdensity` The critical overdensity for collapse of halos, δ_c;

`outputExpansionFactor` The expansion factor;

`growthFactor` The linear growth factor;

`outputRedshift` The redshift;

`outputTime` The cosmic time (in units of Gyr);

`virialDensityContrast` The virial density contrast of halos.

`haloMass` The mass of the halo, M_halo (in $\mathrm{M}_\odot$);

`haloSigma` The root-variance of the mass field smoothed in top-hat spheres, $\sigma(M)$;

`haloAlpha` The loagrithmic gradient of the root-variance of the mass field smoothed in top-hat spheres with mass, $\mathrm{d}\log\sigma(M)/\mathrm{d}\log M_{\rm halo}$;

`haloPeakHeightNu` The peak height of the halo, $\nu = \delta_{\rm c}/\sigma(M)$;

`haloMassFunctionM` The halo mass function per halo mass, $\mathrm{d}n/\mathrm{d}M_{\rm halo}$ (in units of $\mathrm{Mpc}^{-3}\mathrm{M}_\odot^{-1}$);

`haloMassFunctionLnM` The halo mass function per logarithmic halo mass, $\mathrm{d}n/\mathrm{d}\log M_{\rm halo}$ (in units of Mpc^{-3});

`haloMassFunctionLnMBinAveraged` The halo mass function per logarithmic halo mass averaged over the finite width of the bin (in units of Mpc^{-3});

`haloMassFunctionNuFNu` The halo mass function defined in terms of the peak-height parameter, $\nu F(\nu)$;

`haloMassFractionCumulative` The mass fraction in halos above the current halo mass;

`haloMassFunctionCumulative` The cumulative number of halos per unit volume above the current halo mass (in units of Mpc^{-3});

`subhaloMassFunctionCumulative` The cumulative number of sub-halos per unit volume above the current halo mass (in units of Mpc^{-3});

`haloBias` The large scale linear theory bias of the halo;

`haloVirialRadius` The virial radius (in units of Mpc) of the current halo mass;

`haloVirialTemperature` The virial temperature (in units of Kelvin) of the current halo mass;

`haloVirialVelocity` The virial velocity (in units of km/s) of the current halo mass;

`haloScaleRadius` The scale radius (in units of Mpc) of the current halo mass;

`haloVelocityMaximum` The peak of the rotation curve (in units of km/s) of the current halo mass;

Dimensionful datasets have an `unitsInSI` attribute that gives their units in the SI system.

Additionally, an attribute giving the critical density of the universe (in units of $\mathrm{M}_\odot\mathrm{Mpc}^{-3}$) is written to the `cosmology` group and, if such a scale is well-defined, an attribute given the mass corresponding to the scale at which the power spectrum is reduce by half relative to a CDM power spectrum in units of $\mathrm{M}_\odot$ is written as `massHalfMode` to the `powerSpectrum` group.

2.4.4. `excursionSets`

The `excursionSets` task will generate output which contains a variety of measures related to excursion sets in the Press-Schechter formalism. The results are output to a group specified by the `[outputGroup]` subparameter (which defaults to `excursionSets`). The output group contains the following structure:

```
+-> barrier                    [dataset]
|
+-> firstCrossingProbability [dataset]
|
+-> firstCrossingRate         [dataset]
|
+-> mass                       [dataset]
|
```

```
+-> massFunction              [dataset]
|
+-> powerSpectrum             [dataset]
|
+-> time                      [dataset]
|
+-> variance                  [dataset]
|
+-> wavenumber                [dataset]
```

These datasets contain the following information:

`mass` Halo mass [$\mathrm{M}_\odot$];

`time` Cosmic time [Gyr];

`wavenumber` Wavenumber corresponding to this halo mass [Mpc^{-1}];

`powerSpectrum` Power spectrum at this wavenumber [Mpc^3];

`variance` The variance, $S(M) \equiv \sigma^2(M)$, at this halo mass;

`barrier` The excursion set barrier, $B(S)$;

`firstCrossingProbability` The probability of first crossing this barrier between S and $S + \mathrm{d}S$;

`firstCrossingRate` The rate of first crossing of the barrier per unit time [Gyr^{-1}] for all pairs of halo mass;

`massFunction` The halo mass function [$\mathrm{M}_\odot^{-1}$ Mpc^{-3}].

2.4.5. `powerSpectra`

The `powerSpectra` task will output a variety of measures of the matter power spectrum tabulated as a function of wavenumber. The output data has the following structure:

```
+-> powerSpectrum
    |
    +-> alpha          [dataset]
    |
    +-> mass           [dataset]
    |
    +-> powerSpectrum  [dataset]
    |
    +-> sigma          [dataset]
    |
    +-> wavenumber     [dataset]
```

The `Parameters` group contains attributes giving the values of all used parameters (just as in a Galacticus output file). The `powerSpectrum` group contains datasets which give the power spectrum and related properties as follows:

`alpha` The logarithmic slope of $\sigma(M)$: $\alpha = \mathrm{d}\ln\sigma/\mathrm{d}\ln M$;

`mass` The mass scale, M, corresponding to the given wavenumber, k, defined such that $M = 4\pi\Omega_{\rm M}\rho_{\rm crit}/3k^3$ (in units of $\mathrm{M}_\odot$);

`powerSpectrum` The linear theory power spectrum at $z = 0$: $P(k)$ in units of Mpc^3;

`sigma` The dimensionless linear theory mass fluctuation at $z = 0$: $\sigma(M)$;

`wavenumber` The wavenumber in units of Mpc^{-1}.

Dimensionful datasets have an `unitsInSI` attribute that gives their units in the SI system.

3. Input Parameters

The following is an alphnumerically sorted list of all input parameters defined in Galacticus. Each parameter is listed by name, along with a description, default value (if one is specified in Galacticus), the file and program unit with which it is associated. Where relevant, references for parameters and the default values are given.

Name: `A`
Type: real 1
Default value: 2.881 Prada et al. [2011]
Description: The parameter A appearing in the halo concentration algorithm of Prada et al. [2011].
Defined in: `function:prada2011ConstructorParameters`
File: `dark_matter_profiles.structure.concentration.Prada2011.F90`
Used by:

Name: `A1`
Type: real 0..1
Default value: 3.44 [Obreschkow et al., 2009]
Description: The parameter, A_1, appearing in the model for the H_2/HI ratio in galaxies from Obreschkow et al. [2009].
Defined in: `function:obreschkow2009ConstructorParameters`
File: `galacticus.output.analyses.molecular_ratio.Obreschkow2009.F90`
Used by:

Name: `A2`
Type: real 0..1
Default value: 4.82 [Obreschkow et al., 2009]
Description: The parameter, A_2, appearing in the model for the H_2/HI ratio in galaxies from Obreschkow et al. [2009].
Defined in: `function:obreschkow2009ConstructorParameters`
File: `galacticus.output.analyses.molecular_ratio.Obreschkow2009.F90`
Used by:

Name: `B`
Type: real 1
Default value: 1.257 Prada et al. [2011]
Description: The parameter b appearing in the halo concentration algorithm of Prada et al. [2011].
Defined in: `function:prada2011ConstructorParameters`
File: `dark_matter_profiles.structure.concentration.Prada2011.F90`
Used by:

Name: `BCut`
Type: real 1
Default value: 1.47 (Leauthaud et al. 2011; z_1 sample using their `SIG_MOD1` method)
Description: The parameter $B_{\rm cut}$ from the fitting functions of Behroozi et al. [2010].

Defined in: function:behroozi2010ConstructorParameters
File: halo_model.conditional_mass_function.Behroozi2010.F90
Used by:

Name: BSatellite
Type: real 1
Default value: 10.62 (Leauthaud et al. 2011; z_1 sample using their SIG_MOD1 method)
Description: The parameter $B_{\rm sat}$ from the fitting functions of Behroozi et al. [2010].
Defined in: function:behroozi2010ConstructorParameters
File: halo_model.conditional_mass_function.Behroozi2010.F90
Used by:

Name: C
Type: real 1
Default value: 1.022 Prada et al. [2011]
Description: The parameter c appearing in the halo concentration algorithm of Prada et al. [2011].
Defined in: function:prada2011ConstructorParameters
File: dark_matter_profiles.structure.concentration.Prada2011.F90
Used by:

Name: C0
Type: real 1
Default value: 3.681 Prada et al. [2011]
Description: The parameter c_0 appearing in the halo concentration algorithm of Prada et al. [2011].
Defined in: function:prada2011ConstructorParameters
File: dark_matter_profiles.structure.concentration.Prada2011.F90
Used by:

Name: C1
Type: real 1
Default value: 5.033 Prada et al. [2011]
Description: The parameter c_1 appearing in the halo concentration algorithm of Prada et al. [2011].
Defined in: function:prada2011ConstructorParameters
File: dark_matter_profiles.structure.concentration.Prada2011.F90
Used by:

Name: D
Type: real 1
Default value: 0.060 Prada et al. [2011]
Description: The parameter d appearing in the halo concentration algorithm of Prada et al. [2011].
Defined in: function:prada2011ConstructorParameters
File: dark_matter_profiles.structure.concentration.Prada2011.F90
Used by:

Name: F
Type: real 1
Default value: 0.01 Bullock et al. [2001]
Description: The parameter F appearing in the halo concentration algorithm of Bullock et al. [2001].
Defined in: function:bullock2001ConstructorParameters

File: dark_matter_profiles.structure.concentration.Bullock2001.F90
Used by:

Name: G0
Type: real 1
Default value: 0.57
Description: The parameter G_0 appearing in the modified merger rate expression of Parkinson et al. [2008].
Defined in: function:parkinsonColeHellyConstructorParameters
File: merger_trees.branching_probability.Parkinson_Cole_Helly.F90
Used by:

Name: HubbleConstant
Type: real 0..1
Default value: 67.36 (Planck Collaboration et al. 2018; TT,TE,EE+lowE+lensing)
Description: The present day value of the Hubble parameter in units of km/s/Mpc.
Defined in: function:simpleConstructorParameters
File: cosmology.parameters.simple.F90
Used by:

Name: K
Type: real 0..1
Default value: 11.3 [Obreschkow et al., 2009]
Description: The parameter, K (in units of m^4 kg^{-2}), appearing in the model for the H_2/HI ratio in galaxies from Obreschkow et al. [2009].
Defined in: function:obreschkow2009ConstructorParameters
File: galacticus.output.analyses.molecular_ratio.Obreschkow2009.F90
Used by:

Name: OmegaBaryon
Type: real 0..1
Default value: 0.04930 (Planck Collaboration et al. 2018; TT,TE,EE+lowE+lensing)
Description: The density of baryons in the Universe in units of the critical density.
Defined in: function:simpleConstructorParameters
File: cosmology.parameters.simple.F90
Used by:

Name: OmegaDarkEnergy
Type: real 0..1
Default value: 0.6847 (Planck Collaboration et al. 2018; TT,TE,EE+lowE+lensing)
Description: The density of dark energy in the Universe in units of the critical density.
Defined in: function:simpleConstructorParameters
File: cosmology.parameters.simple.F90
Used by:

Name: OmegaMatter
Type: real 0..1
Default value: 0.3153 (Planck Collaboration et al. 2018; TT,TE,EE+lowE+lensing)
Description: The density of matter in the Universe in units of the critical density.

Defined in: function:simpleConstructorParameters
File: cosmology.parameters.simple.F90
Used by:

Name: Rv
Type: real 1
Default value: 2.7
Description: The relative visibility, $R_{\rm V}$.
Defined in: function:prevotBouchetConstructorParameters
File: stellar_populations.spectra.dust_attenuation.Prevot-Bouchet.F90
Used by:

Name: X0
Type: real 1
Default value: 0.424 Prada et al. [2011]
Description: The parameter x_0 appearing in the halo concentration algorithm of Prada et al. [2011].
Defined in: function:prada2011ConstructorParameters
File: dark_matter_profiles.structure.concentration.Prada2011.F90
Used by:

Name: X1
Type: real 1
Default value: 0.526 Prada et al. [2011]
Description: The parameter x_1 appearing in the halo concentration algorithm of Prada et al. [2011].
Defined in: function:prada2011ConstructorParameters
File: dark_matter_profiles.structure.concentration.Prada2011.F90
Used by:

Name: a
Type: real 1
Default value: 1.36 [Trac et al., 2015]
Description: The parameter a for the Tinker et al. [2008] halo mass function.
Defined in: function:tinker2008GenericConstructorParameters
File: structure_formation.halo_mass_function.Tinker2008Generic.F90
Used by:

Name: a1
Type: real 1
Description: Parameter a_1 in the Dutton and Macciò [2014] halo concentration–mass relation.
Defined in: function:duttonMaccio2014ConstructorParameters
File: dark_matter_profiles.structure.concentration.Dutton-Maccio2014.F90
Used by:

Name: a2
Type: real 1
Description: Parameter a_2 in the Dutton and Macciò [2014] halo concentration–mass relation.
Defined in: function:duttonMaccio2014ConstructorParameters
File: dark_matter_profiles.structure.concentration.Dutton-Maccio2014.F90
Used by:

Name: a3
Type: real 1
Description: Parameter a_3 in the Dutton and Macciò [2014] halo concentration–mass relation.
Defined in: function:duttonMaccio2014ConstructorParameters
File: dark_matter_profiles.structure.concentration.Dutton-Maccio2014.F90
Used by:

Name: a4
Type: real 1
Description: Parameter a_4 in the Dutton and Macciò [2014] halo concentration–mass relation.
Defined in: function:duttonMaccio2014ConstructorParameters
File: dark_matter_profiles.structure.concentration.Dutton-Maccio2014.F90
Used by:

Name: absoluteMagnitudeThreshold
Type: real 0..1
Description: The parameter M_0 appearing in the stellar absolute magnitude threshold for the stellar absolute magnitude galactic filter class.
Defined in: function:stellarAbsoluteMagnitudesConstructorParameters
File: galactic.filters.stellar_absolute_magnitudes.F90
Used by:

Name: abundanceMaximum
Type: real 1
Default value: -1.0d0
Description: The abundance (in units of Mpc^{-3}) above which to truncate the halo mass function when sampling halo masses for tree construction. A negative value indicates no truncation.
Defined in: function:haloMassFunctionConstructorParameters
File: merger_trees.construct.build.masses.distribution.halo_mass_function.F90
Used by:

Name: abundanceMinimum
Type: real 1
Default value: -1.0d0
Description: The abundance (in units of Mpc^{-3}) below which to truncate the halo mass function when sampling halo masses for tree construction. A negative value indicates no truncation.
Defined in: function:haloMassFunctionConstructorParameters
File: merger_trees.construct.build.masses.distribution.halo_mass_function.F90
Used by:

Name: accelerationCoefficientGlobal
Type: real 1
Default value: 1.193
Description: Global accleration parameter.
Defined in: function:particleSwarmConstructorParameters
File: posterior_sampling.simulation.particle_swarm.F90
Used by:

Name: `accelerationCoefficientPersonal`
Type: real 1
Default value: 1.193
Description: Personal accleration parameter.
Defined in: `function:particleSwarmConstructorParameters`
File: `posterior_sampling.simulation.particle_swarm.F90`
Used by:

Name: `acceptanceAverageCount`
Type: integer 1
Default value: 10
Description: The number of steps over which to average the acceptance rate.
Defined in: `function:differentialEvolutionConstructorParameters`
File: `posterior_sampling.simulation.differential_evolution.F90`
Used by:

Name: `acceptanceRateMaximum`
Type: real 1
Description: The maximum acceptable acceptance rate.
Defined in: `function:adaptiveConstructorParameters`
File: `posterior_sampling.differential_proposal_size.adaptive.F90`
Used by:

Name: `acceptanceRateMinimum`
Type: real 1
Description: The minimum acceptable acceptance rate.
Defined in: `function:adaptiveConstructorParameters`
File: `posterior_sampling.differential_proposal_size.adaptive.F90`
Used by:

Name: `acceptedStateCount`
Type: integer 1
Default value: 100
Description: The number of states to use in acceptance rate statistics.
Defined in: `function:simpleConstructorParameters`
File: `posterior_sampling.state.simple.F90`
Used by:

Name: `accretionLimit`
Type: real 0..1
Default value: 0.1
Description: The largest fractional mass change due to subresolution accretion allowed in a timestep in merger trees built by the Cole et al. [2000] method.
Defined in: `function:cole2000ConstructorParameters`
File: `merger_trees.construct.builder.Cole2000.F90`
Used by:

Name: `accretionNegativeAllowed`
Type: boolean 1

Default value: .true.
Description: Specifies whether negative accretion (mass loss) is allowed in the simple halo accretion model.
Defined in: function:simpleConstructorParameters
File: accretion.halo.simple.F90
Used by:

Name: accretionNewGrowthOnly
Type: boolean 1
Default value: .false.
Description: Specifies whether accretion from the intergalactic medium (IGM) is allowed only when a halo is growing past its previous greatest mass.
Defined in: function:simpleConstructorParameters
File: accretion.halo.simple.F90
Used by:

Name: accretionRateThinDiskMaximum
Type: real 1
Default value: '0.3d0'
Description: The accretion rate (in Eddington units) above which a switched accretion disk becomes an ADAF.
Defined in: function:switchedConstructorParameters
File: accretion_disks.switched.F90
Used by:

Name: accretionRateThinDiskMinimum
Type: real 1
Default value: '0.01d0'
Description: The accretion rate (in Eddington units) below which a switched accretion disk becomes an ADAF.
Defined in: function:switchedConstructorParameters
File: accretion_disks.switched.F90
Used by:

Name: accretionRateTransitionWidth
Type: real 1
Default value: 0.1
Description: The width (in $\ln[\dot{M}/\dot{M}_{\rm Eddington}]$) over which transitions between accretion disk states occur.
Defined in: function:switchedConstructorParameters
File: accretion_disks.switched.F90
Used by:

Name: accuracyFirstOrder
Type: real 1
Default value: 0.1
Description: Limits the step in $\delta_{\rm crit}$ when constructing merger trees using the Parkinson et al. [2008] algorithm, so that it never exceeds `accuracyFirstOrder` $\sqrt{2[\sigma^2(M_2/2)-\sigma^2(M_2)]}$.
Defined in: function:parkinsonColeHellyConstructorParameters

File: merger_trees.branching_probability.Parkinson_Cole_Helly.F90
Used by:

Name: addHubbleFlow
Type: boolean 0..1
Default value: .false.
Description: If true, Hubble flow will be added to velocity offsets of halos (if applied).
Defined in: function:particulateConstructorParameters
File: merger_trees.operators.particulate.F90
Used by:

Name: adiabaticIndex
Type: real 1
Default value: adafAdiabaticIndexDefault(enumerationAdafFieldEnhancementEncode(char(fieldEnhancementOption),inclu
Description: Specifies the effective adiabatic index of gas in an ADAF.
Defined in: function:adafConstructorParameters
File: accretion_disks.ADAF.F90
Used by:

Name: ageEffective
Type: real 1
Default value: 13.8
Description: The effective age to use for computing SNeIa yield when using the instantaneous stellar evolution approximation.
Defined in: function:standardConstructorParameters
File: stellar_populations.standard.F90
Used by:

Name: ageFactor
Type: real 1
Default value: 0.0
Description: Interpolates (geometrically) between the age of the Universe and the halo dynamical time for the time available for cooling in the White and Frenk [1991] method.
Defined in: function:whiteFrenk1991ConstructorParameters
File: cooling.time_available.White-Frenk.F90
Used by:

Name: ageStatisticsStandardIsInactive
Type: boolean 1
Default value: .false.
Description: Specifies whether or not the variables of the standard age statistics component are inactive (i.e. do not appear in any ODE being solved).
Defined in: subroutine:Node_Component_Age_Statistics_Standard_Initialize
File: objects.nodes.components.age_statistics.standard.F90
Used by:

Name: allTreesExistAtFinalTime
Type: boolean 1
Default value: .true.

Description: Specifies whether or not all merger trees are expected to exist at the final requested output time. If set to false, then trees which finish before a given output time will be ignored.
Defined in: function:standardConstructorParameters
File: merger_trees.evolver.standard.F90
Used by:

Name: allowBranchJumps
Type: boolean 1
Default value: .true.
Description: Specifies whether nodes are allowed to jump between branches.
Defined in: function:readConstructorParameters
File: merger_trees.construct.read.F90
Used by:

Name: allowSubhaloPromotions
Type: boolean 1
Default value: .true.
Description: Specifies whether subhalos are permitted to be promoted to being isolated halos.
Defined in: function:readConstructorParameters
File: merger_trees.construct.read.F90
Used by:

Name: alpha
Type: real 0..1
Default value: 0.0
Description: Parameter α appearing in model for simple systematic shift in the halo mass function.
Defined in: function:simpleSystematicConstructorParameters
File: structure_formation.halo_mass_function.simple_systematic.F90
Used by:

Name: alpha1
Type: real 0..1
Default value: -0.506d0 [Obreschkow et al., 2009]
Description: The parameter, α_1, appearing in the model for the H_2/HI ratio in galaxies from Obreschkow et al. [2009].
Defined in: function:obreschkow2009ConstructorParameters
File: galacticus.output.analyses.molecular_ratio.Obreschkow2009.F90
Used by:

Name: alpha2
Type: real 0..1
Default value: -1.054d0 [Obreschkow et al., 2009]
Description: The parameter, α_2, appearing in the model for the H_2/HI ratio in galaxies from Obreschkow et al. [2009].
Defined in: function:obreschkow2009ConstructorParameters
File: galacticus.output.analyses.molecular_ratio.Obreschkow2009.F90
Used by:

Name: alphaSatellite

Type: real 1
Default value: 1.0 (Leauthaud et al. 2011; z_1 sample using their `SIG_MOD1` method)
Description: The parameter $\alpha_{\rm sat}$ from the fitting functions of Behroozi et al. [2010].
Defined in: `function:behroozi2010ConstructorParameters`
File: `halo_model.conditional_mass_function.Behroozi2010.F90`
Used by:

Name: `alwaysIsolatedHalosOnly`
Type: integer 1
Default value: .true.
Description: Include only always-isolated halos when gathering tree root masses?
Defined in: `function:outputRootMassesConstructorParameters`
File: `merger_trees.operators.output_root_masses.F90`
Used by:

Name: `analyzeAllParticles`
Type: boolean 0..1
Default value: .true.
Description: If true, all particles are assumed to be self-bound at the beginning of the analysis. Unbound particles at previous times are allowed to become bound in the current snapshot. If false and the self-bound information from the previous snapshot is available, only the particles that are self-bound at the previous snapshot are assumed to be bound at the begnning of the anlysis.
Defined in: `function:selfBoundConstructorParameters`
File: `nBody.operator.self_bound.F90`
Used by:

Name: `angleRotation`
Type: float 1
Default value: 2.0d0*Pi*randomSequence%uniformSample() Uniformly random distribution between 0 and 2π.
Description: The angle through which the mock catalog should be rotated.
Defined in: `function:catalogProjectedCorrelationFunctionConstructorParameters`
File: `tasks.catalog_projected_correlation_function.F90`
Used by:

Name: `angularSize`
Type: real 0..1
Description: The angular size (i.e. side length) of the square field of view of the lightcone (in units of degrees).
Defined in: `function:squareConstructorParameters`
File: `geometry.lightcones.square.F90`
Used by:

Name: `apparentMagnitudeThreshold`
Type: real 0..1
Description: The parameter m_0 appearing in the stellar apparent magnitude threshold for the stellar apparent magnitude galactic filter class.
Defined in: `function:stellarApparentMagnitudesConstructorParameters`
File: `galactic.filters.stellar_apparent_magnitudes.F90`

Used by:

Name: `appendLogs`
Type: integer 1
Default value: .false.
Description: If true, do not overwrite existing log files, but instead append to them.
Defined in: `function:differentialEvolutionConstructorParameters`
File: `posterior_sampling.simulation.differential_evolution.F90`
Used by:

Name: `applyTo`
Type: real 0..1
Default value: var_str('nonRates')
Description: Specifies whether rescaling is to be applied to the barrier when used for rate calculation, for other calculations, or both.
Defined in: `function:remapScaleConstructorParameters`
File: `structure_formation.excursion_sets.barrier.remap.scale.F90`
Used by:

Name: `assumeMonotonicSurfaceDensity`
Type: boolean 1
Default value: .false.
Description: If true, assume that the surface density in disks is always monotonically decreasing.
Defined in: `function:krumholz2009ConstructorParameters`
File: `star_formation.rate_surface_density.disks.Krumholz2009.F90`
Used by:

Name: `attemptsMaximum`
Type: logical 0..
Default value: 10000
Description: The maximum allowed number of tree build attempts.
Defined in: `function:augmentConstructorParameters`
File: `merger_trees.operators.augment.F90`
Used by:

Name: `b`
Type: real 1
Default value: 2.54 [Trac et al., 2015]
Description: The parameter b for the Tinker et al. [2008] halo mass function.
Defined in: `function:tinker2008GenericConstructorParameters`
File: `structure_formation.halo_mass_function.Tinker2008Generic.F90`
Used by:

Name: `b1`
Type: real 1
Description: Parameter b_1 in the Dutton and Macciò [2014] halo concentration–mass relation.
Defined in: `function:duttonMaccio2014ConstructorParameters`
File: `dark_matter_profiles.structure.concentration.Dutton-Maccio2014.F90`
Used by:

Name: b2
Type: real 1
Description: Parameter b_2 in the Dutton and Macciò [2014] halo concentration–mass relation.
Defined in: function:duttonMaccio2014ConstructorParameters
File: dark_matter_profiles.structure.concentration.Dutton-Maccio2014.F90
Used by:

Name: bRatioHigh
Type: real 1
Default value: [+2.878d0,+3.946d0,+2.982d0]
Description: Values of the B parameter of the Jiang et al. [2014] orbital velocity distribution for the three host halo mass ranges, and the 0.05–0.5 mass ratio.
Defined in: function:jiang2014ConstructorParameters
File: satellites.merging.virial_orbits.Jiang2014.F90
Used by:

Name: bRatioIntermediate
Type: real 1
Default value: [+1.044d0,+1.535d0,+3.396d0]
Description: Values of the B parameter of the Jiang et al. [2014] orbital velocity distribution for the three host halo mass ranges, and the 0.005–0.05 mass ratio.
Defined in: function:jiang2014ConstructorParameters
File: satellites.merging.virial_orbits.Jiang2014.F90
Used by:

Name: bRatioLow
Type: real 1
Default value: [+0.049d0,+0.548d0,+1.229d0]
Description: Values of the B parameter of the Jiang et al. [2014] orbital velocity distribution for the three host halo mass ranges, and the 0.0001–0.005 mass ratio.
Defined in: function:jiang2014ConstructorParameters
File: satellites.merging.virial_orbits.Jiang2014.F90
Used by:

Name: badValue
Type: real 1
Default value: -0.5d0
Description: Use for bad value detection in "Sussing" merger trees. Values for scale radius and halo spin which exceed this threshold are assumed to be bad.
Defined in: function:sussingConstructorParameters
File: merger_trees.construct.read.importer.SussingMergerTrees.F90
Used by:

Name: badValueTest
Type: real 1
Default value: var_str('lessThan')
Description: Use for bad value detection in "Sussing" merger trees. Values which exceed the threshold in ths specified direction are assumed to be bad.

Defined in: function:sussingConstructorParameters
File: merger_trees.construct.read.importer.SussingMergerTrees.F90
Used by:

Name: band
Type: string 0..1
Description: The band (u, g, r, i, or z) for which the luminosity function should be computed.
Defined in: function:luminosityFunctionMonteroDorta2009SDSSConstructorParameters
File: galacticus.output.analyses.luminosity_function.Montero-Dorta2009.SDSS.F90
Used by:

Name: baseMassMaximum
Type: real 0..1
Default value: 0.0
Description: Base node mass above which trees should be ignored.
Defined in: function:selectWithinRangeConstructorParameters
File: merger_trees.operators.select_within_range.F90
Used by:

Name: baseMassMinimum
Type: real 0..1
Default value: 0.0
Description: Base node mass below which trees should be ignored.
Defined in: function:selectWithinRangeConstructorParameters
File: merger_trees.operators.select_within_range.F90
Used by:

Name: baseParametersFileName
Type: string 1
Description: The base set of parameters to use.
Defined in: function:galaxyPopulationConstructorParameters
File: models.likelihoods.galaxy_population.F90
Used by:

Name: beginAt
Type: integer 1
Default value: -1_kind_int8
Description: Specifies the index of the tree to begin at. (Use -1 to always begin with the first tree.)
Defined in: function:readConstructorParameters
File: merger_trees.construct.read.F90
Used by:

Name: beta
Type: real 0..1
Default value: 0.0
Description: Parameter β appearing in model for simple systematic shift in the halo mass function.
Defined in: function:simpleSystematicConstructorParameters
File: structure_formation.halo_mass_function.simple_systematic.F90
Used by:

Name: `beta0`
Type: real 0..1
Default value: 13.0 [Creasey et al., 2012]
Description: The parameter β_0 appearing in the Creasey et al. [2012] model for supernovae feedback.
Defined in: `function:creasey2012ConstructorParameters`
File: `star_formation.feedback.disks.Creasey2012.F90`
Used by:

Name: `betaCut`
Type: real 1
Default value: -0.13d0 (Leauthaud et al. 2011; z_1 sample using their `SIG_MOD1` method)
Description: The parameter $\beta_{\rm cut}$ from the fitting functions of Behroozi et al. [2010].
Defined in: `function:behroozi2010ConstructorParameters`
File: `halo_model.conditional_mass_function.Behroozi2010.F90`
Used by:

Name: `betaSatellite`
Type: real 1
Default value: 0.859 (Leauthaud et al. 2011; z_1 sample using their `SIG_MOD1` method)
Description: The parameter $\beta_{\rm sat}$ from the fitting functions of Behroozi et al. [2010].
Defined in: `function:behroozi2010ConstructorParameters`
File: `halo_model.conditional_mass_function.Behroozi2010.F90`
Used by:

Name: `binCenter`
Type: float 0..1
Description: The value of the property at the center of each bin.
Defined in: `function:volumeFunction1DConstructorParameters`
File: `galacticus.output.analyses.volume_function_1d.F90`
Used by:

Name: `binCountMinimum`
Type: real 1
Description: The minimum number of halos per bin required to permit bin to be included in likelihood evaluation.
Defined in: `function:haloMassFunctionConstructorParameters`
File: `models.likelihoods.halo_mass_function.F90`
Used by:

Name: `binWidthLogarithmic`
Type: real 1
Description: The logarithmic width of bins in the stellar mass function to be assumed when computing the optimal sampling density function for tree masses.
Defined in: `function:stellarMassFunctionConstructorParameters`
File: `merger_trees.construct.build.masses.distribution.stellar_mass_function.F90`
Used by:

Name: `binaryFormatOld`

Type: boolean 1
Default value: .false.
Description: Specifies whether the old binary format is to be used (for reading only).
Defined in: function:sussingASCIIConstructorParameters
File: merger_trees.construct.read.importer.SussingMergerTrees.ASCII.F90
Used by:

Name: binnedProjectedCorrelationCovarianceTarget
Type: real 0..1
Description: The target function covariance for likelihood calculations.
Defined in: function:correlationFunctionConstructorParameters
File: galacticus.output.analyses.correlation_function.F90
Used by:

Name: binnedProjectedCorrelationTarget
Type: real 0..1
Description: The target function for likelihood calculations.
Defined in: function:correlationFunctionConstructorParameters
File: galacticus.output.analyses.correlation_function.F90
Used by:

Name: birthCloudLifetime
Type: real 1
Default value: 1.0×10^{-2}
Description: The duration which stars remain within their birth clouds.
Defined in: function:charlotFall2000ConstructorParameters
File: stellar_populations.spectra.dust_attenuation.CharlotFall2000.F90
Used by:

Name: blackHoleHeatingEfficiency
Type: double 1
Default value: 1.0×10^{-3}
Description: The efficiency with which accretion onto a black hole heats the hot halo.
Defined in: subroutine:Node_Component_Black_Hole_Simple_Initialize
File: objects.nodes.components.black_hole.simple.F90
Used by:

Name: blackHoleHeatsHotHalo
Type: boolean 1
Default value: .true.
Description: Specifies whether or not the black hole launched jets should heat the hot halo.
Defined in: subroutine:Node_Component_Black_Hole_Standard_Initialize
File: objects.nodes.components.black_hole.standard.F90
Used by:

Name: blackHoleOutputAccretion
Type: boolean 1
Default value: .false.
Description: Determines whether or not accretion rates and jet powers will be output.

Defined in: subroutine:Node_Component_Black_Hole_Standard_Initialize
File: objects.nodes.components.black_hole.standard.F90
Used by:

Name: blackHoleOutputData
Type: boolean 1
Default value: .false.
Description: Determines whether or not properties for all black holes (rather than just the central black hole) will be output.
Defined in: subroutine:Node_Component_Black_Hole_Standard_Initialize
File: objects.nodes.components.black_hole.standard.F90
Used by:

Name: blackHoleOutputMergers
Type: boolean 1
Default value: .false.
Description: Determines whether or not properties of black hole mergers will be output.
Defined in: subroutine:Node_Component_Black_Hole_Standard_Initialize
File: objects.nodes.components.black_hole.standard.F90
Used by:

Name: blackHoleRadioModeFeedbackEfficiency
Type: double 1
Default value: 1.0
Description: Efficiency with which radio-mode feedback is coupled to the hot halo.
Defined in: subroutine:Node_Component_Black_Hole_Standard_Initialize
File: objects.nodes.components.black_hole.standard.F90
Used by:

Name: blackHoleSeedMass
Type: real 1
Default value: 100.0
Description: The mass of the seed black hole placed at the center of each newly formed galaxy.
Defined in: function:Node_Component_Black_Hole_Simple_Seed_Mass
File: objects.nodes.components.black_hole.simple.bound_functions.Inc
Used by:

Name: blackHoleToSpheroidStellarGrowthRatio
Type: double 1
Default value: 1.0×10^{-3}
Description: The ratio of the rates of black hole growth and spheroid stellar mass growth.
Defined in: subroutine:Node_Component_Black_Hole_Simple_Initialize
File: objects.nodes.components.black_hole.simple.F90
Used by:

Name: blackHoleWindEfficiency
Type: double 1
Default value: 2.4×10^{-3}
Description: The efficiency of the black hole-driven wind: $L_{\rm wind} = \epsilon_{\rm wind} \dot{M}_\bullet c^2$.

Defined in: subroutine:Node_Component_Black_Hole_Standard_Initialize
File: objects.nodes.components.black_hole.standard.F90
Used by:

Name: blackHoleWindEfficiencyScalesWithRadiativeEfficiency
Type: double 1
Default value: .false.
Description: Specifies whether the black hole wind efficiency should scale with the radiative efficiency of the accretion disk.
Defined in: subroutine:Node_Component_Black_Hole_Standard_Initialize
File: objects.nodes.components.black_hole.standard.F90
Used by:

Name: bondiHoyleAccretionEnhancementHotHalo
Type: double 1
Default value: 6.0
Description: The factor by which the Bondi-Hoyle accretion rate of hot halo gas onto black holes in enhanced.
Defined in: subroutine:Node_Component_Black_Hole_Standard_Initialize
File: objects.nodes.components.black_hole.standard.F90
Used by:

Name: bondiHoyleAccretionEnhancementSpheroid
Type: double 1
Default value: 5.0
Description: The factor by which the Bondi-Hoyle accretion rate of spheroid gas onto black holes in enhanced.
Defined in: subroutine:Node_Component_Black_Hole_Standard_Initialize
File: objects.nodes.components.black_hole.standard.F90
Used by:

Name: bondiHoyleAccretionHotModeOnly
Type: double 1
Default value: .true.
Description: Determines whether accretion from the hot halo should only occur if the halo is in the hot accretion mode.
Defined in: subroutine:Node_Component_Black_Hole_Standard_Initialize
File: objects.nodes.components.black_hole.standard.F90
Used by:

Name: bondiHoyleAccretionTemperatureSpheroid
Type: double 1
Default value: 1.0×10^2
Description: The assumed temperature (in Kelvin) of gas in the spheroid when computing Bondi-Hoyle accretion rates onto black holes.
Defined in: subroutine:Node_Component_Black_Hole_Standard_Initialize
File: objects.nodes.components.black_hole.standard.F90
Used by:

Name: bootstrapSampleCount
Type: integer 0..1
Default value: 30_c_size_t
Description: The number of bootstrap resamples of the particles that should be used.
Defined in: function:velocityDispersionConstructorParameters
File: nBody.operator.velocity_dispersion.F90
Used by:

Name: bootstrapSampleRate
Type: float 0..1
Default value: 1.0
Description: The sampling rate for particles.
Defined in: function:selfBoundConstructorParameters
File: nBody.operator.self_bound.F90
Used by:

Name: branchIntervalStep
Type: boolean 0..1
Default value: .true.
Description: If `false` use the original Cole et al. [2000] method to determine whether branching occurs in a timestep. If `true` draw branching intervals from a negative exponential distribution.
Defined in: function:cole2000ConstructorParameters
File: merger_trees.construct.builder.Cole2000.F90
Used by:

Name: bufferCount
Type: integer 0..1
Description: The number of buffer bins to include below and above the range of actual bins.
Defined in: function:volumeFunction1DConstructorParameters
File: galacticus.output.analyses.volume_function_1d.F90
Used by:

Name: bufferIsolatedHalos
Type: boolean 0..1
Default value: .false.
Description: If true, intersection of a tree with the lightcone will be determined using the positions non-isolated (a.k.a. "satellite") halos, and of isolated halos (a.k.a "centrals") with a buffer region (with radius equal to the extent of the orphan satellite distribution—see §16.4.1) placed around each such halo, and any intersection of that region with the lightcone is sufficient to prevent pruning of the tree. If this parameter is `false` then (unbuffered) positions of all halos are used for determining intersection with the lightcone—this requires complete (i.e. throughout the extent of their existance) knowledge of non-isolated halos prior to application of this operator.
Defined in: function:pruneLightconeConstructorParameters
File: merger_trees.operators.prune_lightcone.F90
Used by:

Name: burnCount
Type: integer 1
Default value: 0

Description: The number of steps to burn before computing convergence.
Defined in: function:gelmanRubinConstructorParameters
File: posterior_sampling.convergence.Gelman-Rubin.F90
Used by:

Name: burstCount
Type: real 1
Description: The number of bursts events to include in the star formation history.
Defined in: function:sedFitConstructorParameters
File: models.likelihoods.SED_fit.F90
Used by:

Name: c
Type: real 1
Default value: 1.14 [Trac et al., 2015]
Description: The parameter c for the Tinker et al. [2008] halo mass function.
Defined in: function:tinker2008GenericConstructorParameters
File: structure_formation.halo_mass_function.Tinker2008Generic.F90
Used by:

Name: cdmAssumptions
Type: real 1
Default value: .false.
Description: If true, assume that $\alpha(= -\mathrm{d}\log\sigma/\mathrm{d}\log M) > 0$ and $\mathrm{d}\alpha/\mathrm{d}M > 0$ (as is true in the case of CDM) when constructing merger trees using the Parkinson et al. [2008].
Defined in: function:parkinsonColeHellyConstructorParameters
File: merger_trees.branching_probability.Parkinson_Cole_Helly.F90
Used by:

Name: chainBaseName
Type: string 1
Description: The base name of the MCMC chain files to read.
Defined in: function:posteriorAsPriorConstructorParameters
File: models.likelihoods.posterior_as_prior.F90
Used by:

Name: chemicalsToTrack
Type: string 1..*
Description: The names of the chemicals to be tracked.
Defined in: subroutine:Chemical_Abundances_Initialize
File: objects.chemical_abundances.F90
Used by:

Name: chunkSize
Type: integer 0..1
Default value: -1
Description: HDF5 dataset chunk size.
Defined in: function:particulateConstructorParameters
File: merger_trees.operators.particulate.F90

Used by:

Name: `clumpingFactorMolecularComplex`
Type: real 1
Default value: 5.0
Description: The density enhancement (relative to mean disk density) for molecular complexes in the "Krumholz-McKee-Tumlinson" star formation timescale calculation.
Defined in: `function:krumholz2009ConstructorParameters`
File: `star_formation.rate_surface_density.disks.Krumholz2009.F90`
Used by:

Name: `coefficient`
Type: float 0..1
Description: The coefficients in the polynomial systematic offset property operator class.
Defined in: `function:systmtcPolynomialConstructorParameters`
File: `galacticus.output.analyses.property_operator.systematic.polynomial.F90`
Used by:

Name: `coefficientConstant`
Type: real 0..1
Default value: 1.67
Description: The constant coefficient in the quadratic excursion set barrier.
Defined in: `function:quadraticConstructorParameters`
File: `structure_formation.excursion_sets.barrier.quadratic.F90`
Used by:

Name: `coefficientLinear`
Type: real 0..1
Default value: 0.0
Description: The linear coefficient in the quadratic excursion set barrier.
Defined in: `function:quadraticConstructorParameters`
File: `structure_formation.excursion_sets.barrier.quadratic.F90`
Used by:

Name: `coefficientQuadratic`
Type: real 0..1
Default value: 0.0
Description: The quadratic coefficient in the quadratic excursion set barrier.
Defined in: `function:quadraticConstructorParameters`
File: `structure_formation.excursion_sets.barrier.quadratic.F90`
Used by:

Name: `column`
Type: integer 1
Description: The column from which to read the particle property.
Defined in: `function:mergerTreeFileBuilderConstructorParameters`
File: `tasks.merger_tree_file_builder.F90`
Used by:

Name: `columnHeaders`
Type: string 1
Default value: .false.
Description: If true, the file is assumed to contain a single line of column headers, which will be skipped.
Defined in: function:mergerTreeFileBuilderConstructorParameters
File: tasks.merger_tree_file_builder.F90
Used by:

Name: `columnSeparator`
Type: string 1
Default value: var_str(',')
Description: The separator for columns.
Defined in: function:mergerTreeFileBuilderConstructorParameters
File: tasks.merger_tree_file_builder.F90
Used by:

Name: `comment`
Type: string 0..1
Description: A descriptive comment for the analysis.
Defined in: function:volumeFunction1DConstructorParameters
File: galacticus.output.analyses.volume_function_1d.F90
Used by:

Name: `computeCovariances`
Type: boolean 1
Default value: .false.
Description: Compute covariances for accumulated statistics?
Defined in: function:conditionalMFConstructorParameters
File: merger_trees.operators.conditional_mass_function.F90
Used by:

Name: `computeScatter`
Type: logical 0..1
Default value: .false.
Description: If true, the scatter in log10(stellar mass) is computed. Otherwise, the mean is computed.
Defined in: function:stellarVsHaloMassRelationLeauthaud2012ConstructorParameters
File: galacticus.output.analyses.stellar_vs_halo_mass_relation.COSMOS_Leauthaud2012.F90
Used by:

Name: `computeVelocityDispersion`
Type: boolean 1
Default value: .false.
Description: Specifies whether or not the velocity dispersion of dark matter and stars should be computed using Jeans equation in black hole binary hardening calculations. If `false`, then the velocity dispersions are assumed to equal the characteristic velocity of dark matter and spheroid.
Defined in: function:standardConstructorParameters
File: black_holes.binaries.separation_growth_rate.standard.F90
Used by:

Name: `concentration`
Type: real 1
Default value: 1.0
Description: The concentration of the NFW profile.
Defined in: `function:nfwConstructorParameters`
File: `mass_distributions.spherical.NFW.F90`
Used by:

Name: `constant`
Type: real 1
Description: The constant error contribution to the stellar mass function to be assumed when computing the optimal sampling density function for tree masses.
Defined in: `function:stellarMassFunctionConstructorParameters`
File: `merger_trees.construct.build.masses.distribution.stellar_mass_function.F90`
Used by:

Name: `content`
Type: string 1
Description: The value of the metadata.
Defined in: `function:mergerTreeFileBuilderConstructorParameters`
File: `tasks.merger_tree_file_builder.F90`
Used by:

Name: `conversionFactor`
Type: float 1
Description: An additional conversion factor to apply to the property to get it into the correct units.
Defined in: `function:mergerTreeFileBuilderConstructorParameters`
File: `tasks.merger_tree_file_builder.F90`
Used by:

Name: `convertToBinary`
Type: boolean 1
Default value: .true.
Description: Specifies whether halo and tree files in the "Sussing" format should be converted to binary the first time they are read and stored to file. This allows rapid re-reading in future.
Defined in: `function:sussingASCIIConstructorParameters`
File: `merger_trees.construct.read.importer.SussingMergerTrees.ASCII.F90`
Used by:

Name: `coreRadius`
Type: real 1
Description: The core radius of a β-model mass distribution.
Defined in: `function:betaProfileConstructorParameters`
File: `mass_distributions.spherical.beta_profile.F90`
Used by:

Name: `coreRadiusOverScaleRadius`
Type: real 1
Default value: 0.1

Description: The core radius in the hot halo density profile in units of the dark matter profile scale radius.
Defined in: function:growingConstructorParameters
File: hot_halo.mass_distribution.cored.core_radius.growing.F90
Used by:

Name: coreRadiusOverVirialRadius
Type: real 1
Default value: 0.3
Description: The core radius in the hot halo density profile in units of the virial radius.
Defined in: function:virialFractionConstructorParameters
File: hot_halo.mass_distribution.cored.core_radius.virial_radius_fraction.F90
Used by:

Name: coreRadiusOverVirialRadiusMaximum
Type: real 1
Default value: 10.0
Description: The maximum core radius in the "growing" hot halo density profile in units of the virial radius.
Defined in: function:growingConstructorParameters
File: hot_halo.mass_distribution.cored.core_radius.growing.F90
Used by:

Name: correctForConcentrationDefinition
Type: string 1
Default value: .false.
Description: If true, then when computing dark matter profile scale radii using concentrations, any difference between the current definition of halo scales (i.e. typically virial density contrast definitions) and density profiles and those assumed in measuring the concentrations will be taken into account. If false, the concentration is applied blindly.
Defined in: function:concentrationConstructorParameters
File: dark_matter_profiles.structure.scale.concentration.F90
Used by:

Name: correlationMassExponent
Type: real 0..1
Default value: 0.0
Description: Variable α in the model for the correlation between halo mass errors: $C_{12} = C_0[M_2/M_1]^\alpha[(1+z_2)/(1+z_1)]^\beta$.
Defined in: function:nbodyHaloMassErrorTrenti2010Parameters
File: statistics.Nbody.halos.mass_errors.Trenti2010.F90
Used by:

Name: correlationNormalization
Type: real 0..1
Default value: 0.0
Description: Variable C_0 in the model for the correlation between halo mass errors: $C_{12} = C_0[M_2/M_1]^\alpha[(1+z_2)/(1+z_1)]^\beta$.
Defined in: function:nbodyHaloMassErrorTrenti2010Parameters

File: statistics.Nbody.halos.mass_errors.Trenti2010.F90
Used by:

Name: correlationRedshiftExponent
Type: real 0..1
Default value: 0.0
Description: Variable β in the model for the correlation between halo mass errors: $C_{12} = C_0[M_2/M_1]^{\alpha}[(1+z_2)/(1+z_1)]^{\beta}$.
Defined in: function:nbodyHaloMassErrorTrenti2010Parameters
File: statistics.Nbody.halos.mass_errors.Trenti2010.F90
Used by:

Name: countFailures
Type: real 1
Description: The number of failures.
Defined in: function:negativeBinomialConstructorParameters
File: statistics.distributions.discrete.negative_binomial.F90
Used by:

Name: countHistoryTimes
Type: integer 1
Default value: 10
Description: The number of times at which a galaxy's stellar properties history is stored.
Defined in: function:noninstantaneousConstructorParameters
File: stellar_populations.properties.noninstantaneous.F90
Used by:

Name: countMass
Type: string 1
Default value: 21
Description: The number of bins for which to compute the conditional mass function.
Defined in: function:conditionalMassFunctionConstructorParameters
File: tasks.conditional_mass_function.F90
Used by:

Name: countMassBins
Type: boolean 1
Default value: 10
Description: The number of bins in the mass function for covariance calculations.
Defined in: function:massFunctionCovarianceConstructorParameters
File: tasks.mass_function_covariance.F90
Used by:

Name: countMetallicities
Type: integer 1
Default value: 10
Description: The number of bins in metallicity to use when tabulating star formation histories.
Defined in: function:metallicitySplitConstructorParameters
File: star_formation.histories.metallicity_split.F90

Used by:

Name: `countSeparations`
Type: string 1
Description: The number of separations at which to compute the projected correlation function.
Defined in: `function:haloModelProjectedCorrelationFunctionConstructorParameters`
File: `tasks.halo_model.projected_correlation_function.F90`
Used by:

Name: `countSteps`
Type: integer 1
Default value: 100
Description: The number of steps (spaced logarithmically in cosmic time) at which to tabulate the evolution of main branch progenitor galaxies.
Defined in: `function:recordEvolutionConstructorParameters`
File: `merger_trees.evolve.timesteps.record_evolution.F90`
Used by:

Name: `countTrials`
Type: integer 1
Description: The number of trials.
Defined in: `function:binomialConstructorParameters`
File: `statistics.distributions.discrete.binomial.F90`
Used by:

Name: `covariance`
Type: real 1
Description: The covariance matrix for the of the multivariate normal distribution.
Defined in: `function:multivariateNormalStochasticConstructorParameters`
File: `models.likelihoods.multivariate_normal.stochastic.F90`
Used by:

Name: `covarianceBinomialBinsPerDecade`
Type: real 0..1
Default value: 10
Description: The number of bins per decade of halo mass to use when constructing volume function covariance matrices for main branch galaxies.
Defined in: `function:volumeFunction1DConstructorParameters`
File: `galacticus.output.analyses.volume_function_1d.F90`
Used by:

Name: `covarianceBinomialMassHaloMaximum`
Type: real 0..1
Default value: 1.0×10^{16}
Description: The maximum halo mass to consider when constructing volume function covariance matrices for main branch galaxies.
Defined in: `function:volumeFunction1DConstructorParameters`
File: `galacticus.output.analyses.volume_function_1d.F90`
Used by:

Name: `covarianceBinomialMassHaloMinimum`
Type: real 0..1
Default value: 1.0×10^8
Description: The minimum halo mass to consider when constructing volume function covariance matrices for main branch galaxies.
Defined in: function:volumeFunction1DConstructorParameters
File: galacticus.output.analyses.volume_function_1d.F90
Used by:

Name: `covarianceModel`
Type: float 0..1
Description: The model to use for computing covariances.
Defined in: function:volumeFunction1DConstructorParameters
File: galacticus.output.analyses.volume_function_1d.F90
Used by:

Name: `cpuLimit`
Type: integer 1
Default value: 0_c_long
Description: The CPU time limit for the run, in seconds.
Defined in: subroutine:System_Limits_Set
File: system.limits.F90
Used by:

Name: `criticalOverdensity`
Type: real 0..1
Default value: (3.0d0/20.0d0)*(12.0d0*Pi)**(2.0d0/3.0d0)
Description: The value to use for the critical overdensity for collapse of dark matter halos when using a fixed value.
Defined in: function:fixedConstructorParameters
File: structure_formation.critical_overdensity.fixed.F90
Used by:

Name: `darkEnergyEquationOfStateW0`
Type: real 1
Default value: -1.0d0
Description: The equation of state parameter for dark energy, w_0, defined such that $P = \rho^w$ with $w(a) = w_0 + w_1 a(1-a)$.
Defined in: function:matterDarkEnergyConstructorParameters
File: cosmology.functions.matter_dark_energy.F90
Used by:

Name: `darkEnergyEquationOfStateW1`
Type: real 1
Default value: 0.0
Description: The equation of state parameter for dark energy, w_1, defined such that $P = \rho^w$ with $w(a) = w_0 + w_1 a(1-a)$.
Defined in: function:matterDarkEnergyConstructorParameters

File: cosmology.functions.matter_dark_energy.F90
Used by:

Name: darkMatterOnly
Type: boolean 1
Default value: .false.
Description: Specifies whether or not density contrast data should be computed using the dark matter component alone.
Defined in: function:densityContrastsConstructorParameters
File: nodes.property_extractor.density_contrasts.F90
Used by:

Name: darkMatterProfileMaximumConcentration
Type: double 1
Default value: 100.0
Description: The maximum concentration allowed for dark matter profiles.
Defined in: subroutine:Node_Component_Dark_Matter_Profile_Scale_Initialize
File: objects.nodes.components.dark_matter_profile.scale.F90
Used by:

Name: darkMatterProfileMinimumConcentration
Type: double 1
Default value: 4.0
Description: The minimum concentration allowed for dark matter profiles.
Defined in: subroutine:Node_Component_Dark_Matter_Profile_Scale_Initialize
File: objects.nodes.components.dark_matter_profile.scale.F90
Used by:

Name: definition
Type: string 1
Description: The definition of the parameter.
Defined in: function:derivedConstructorParameters
File: models.parameters.derived.F90
Used by:

Name: degreesOfFreedom
Type: real 1
Description: The degrees of freedom of the Student-t distribution function.
Defined in: function:studentTConstructorParameters
File: statistics.distributions.Student-t.F90
Used by:

Name: degreesOfFreedomEffective
Type: real 1
Default value: 1.5
Description: The effective number of degrees of freedom for the thermal warm dark matter particle.
Defined in: function:wdmThermalConstructorParameters
File: dark_matter.particle.warm_dark_matter.thermal.F90
Used by:

Name: `delta`
Type: real 1
Default value: 0.5666 (Leauthaud et al. 2011; z_1 sample using their `SIG_MOD1` method)
Description: The parameter δ from the fitting functions of Behroozi et al. [2010].
Defined in: `function:behroozi2010ConstructorParameters`
File: `halo_model.conditional_mass_function.Behroozi2010.F90`
Used by:

Name: `delta1`
Type: real 1
Default value: 2.40 Henriques et al. [2013]
Description: The exponent of the $(1 + z)$ term, δ_1.
Defined in: `function:henriques2013ConstructorParameters`
File: `hot_halo.outflow_reincorporation.Henriques2013.F90`
Used by:

Name: `delta2`
Type: real 1
Default value: 3.07 Henriques et al. [2013]
Description: The exponent of the $V_{\rm vir}$ term, δ_2.
Defined in: `function:henriques2013ConstructorParameters`
File: `hot_halo.outflow_reincorporation.Henriques2013.F90`
Used by:

Name: `deltaStepMaximum`
Type: real 1
Default value: 0.1
Description: Limits the step in $\delta_{\rm crit}$ when constructing merger trees using the generalized Press-Schechter branching algorithm.
Defined in: `function:generalizedPressSchechterConstructorParameters`
File: `merger_trees.branching_probability.generalized_Press_Schechter.F90`
Used by:

Name: `densityContrastValue`
Type: float 1
Default value: 200.0
Description: The virial density contrast to use in the fixed value model.
Defined in: `function:fixedConstructorParameters`
File: `structure_formation.virial_density_contrast.fixed.F90`
Used by:

Name: `densityContrasts`
Type: real 1..*
Description: A list of density contrasts at which to output data.
Defined in: `function:densityContrastsConstructorParameters`
File: `nodes.property_extractor.density_contrasts.F90`
Used by:

Name: densityNormalization
Type: real 1
Description: The density normalization of a β-model mass distribution.
Defined in: function:betaProfileConstructorParameters
File: mass_distributions.spherical.beta_profile.F90
Used by:

Name: densityParticleMean
Type: real 0..1
Description: The mean density of particles in the simulation.
Defined in: function:environmentalOverdensityConstructorParameters
File: nBody.operator.environmental_overdensity.F90
Used by:

Name: densityRatio
Type: real 1
Default value: 4.688 Value appropriate for an Navarro-Frenk-White (dark matter halo profile) (NFW) profile with concentration $c = 6.88$ which is the concentration found by Prada et al. [2011] for halos with $\sigma = 1.686$ which is the approximate critical overdensity for collapse.
Description: The ratio of mean virial density to density at the virial radius to assume when setting virial density contrasts in the friends-of-friends model.
Defined in: function:friendsOfFriendsConstructorParameters
File: structure_formation.virial_density_contrast.friends-of-friends.F90
Used by:

Name: densityType
Type: string 1
Default value: var_str('critical')
Description: The reference density to use in the fixed value virial density contrast model. Either of critical and mean are allowed.
Defined in: function:fixedConstructorParameters
File: structure_formation.virial_density_contrast.fixed.F90
Used by:

Name: depthLineOfSight
Type: string 1
Description: The maximum line of sight depth to which to integrate when computing the projected correlation function.
Defined in: function:haloModelProjectedCorrelationFunctionConstructorParameters
File: tasks.halo_model.projected_correlation_function.F90
Used by:

Name: depthOpticalCloudsCoefficient
Type: string 0..1
Default value: 1.0
Description: Multiplicative coefficient for optical depth in birth clouds.
Defined in: function:lmnstyStllrChrltFll2000ConstructorParameters
File: nodes.property_extractor.luminosity_stellar.dust.CharlotFall2000.F90
Used by:

Name: `depthOpticalISMCoefficient`
Type: string 0..1
Default value: 1.0
Description: Multiplicative coefficient for optical depth in the ISM.
Defined in: `function:lmnstyStllrChrltFll2000ConstructorParameters`
File: `nodes.property_extractor.luminosity_stellar.dust.CharlotFall2000.F90`
Used by:

Name: `description`
Type: boolean 1
Description: A description of this property.
Defined in: `function:ratioConstructorParameters`
File: `nodes.property_extractor.ratio.F90`
Used by:

Name: `destinationGasMinorMerger`
Type: string 1
Default value: var_str('spheroid')
Description: The component to which satellite galaxy gas moves to as a result of a minor merger.
Defined in: `function:simpleConstructorParameters`
File: `satellites.merging.mass_movements.simple.F90`
Used by:

Name: `dimensionless`
Type: boolean 1
Description: If true then the β-model mass distribution is considered to be in dimensionless units.
Defined in: `function:betaProfileConstructorParameters`
File: `mass_distributions.spherical.beta_profile.F90`
Used by:

Name: `discriminator`
Type: string 1
Default value: var_str('boundMass')
Description: Specifies whether bound mass or position history will be used to determine satellite orphan status.
Defined in: `function:satelliteStatusConstructorParameters`
File: `nodes.property_extractor.satellite_status.F90`
Used by:

Name: `diskLuminositiesStellarInactive`
Type: boolean 1
Default value: .false.
Description: Specifies whether or not disk stellar luminosities are inactive properties (i.e. do not appear in any ODE being solved).
Defined in: `subroutine:Node_Component_Disk_Standard_Initialize`
File: `objects.nodes.components.disk.standard.F90`
Used by:

Name: diskMassToleranceAbsolute
Type: double 1
Default value: 1.0×10^{-6}
Description: The mass tolerance used to judge whether the disk is physically plausible.
Defined in: subroutine:Node_Component_Disk_Very_Simple_Size_Initialize
File: objects.nodes.components.disk.very_simple.size.F90
Used by:

Name: diskNegativeAngularMomentumAllowed
Type: double 1
Default value: .true.
Description: Specifies whether or not negative angular momentum is allowed for the disk.
Defined in: subroutine:Node_Component_Disk_Standard_Initialize
File: objects.nodes.components.disk.standard.F90
Used by:

Name: diskOutflowTimescaleMinimum
Type: double 1
Default value: 1.0×10^{-3}
Description: The minimum timescale (in units of the halo dynamical time) on which outflows may deplete gas in the disk.
Defined in: subroutine:Node_Component_Disk_Very_Simple_Initialize
File: objects.nodes.components.disk.very_simple.F90
Used by:

Name: diskOutputStarFormationRate
Type: string 1
Default value: .false.
Description: Specifies whether the
normalfont
ttfamily starFormationRate method of the
normalfont
ttfamily standard implemention of the
normalfont
ttfamily component class should be output.
Defined in: subroutine:nodeClassHierarchyInitialize
File: objects.nodes.components.Inc
Used by:

Name: diskRadiusSolverCole2000Method
Type: boolean 1
Default value: .false.
Description: HASH(0x3169f28)
Defined in: subroutine:Node_Component_Disk_Standard_Initialize
File: objects.nodes.components.disk.standard.F90
Used by:

Name: diskStarFormationInSatellites
Type: boolean 1

Default value: .true.
Description: Specifies whether or not star formation occurs in disks in satellites.
Defined in: subroutine:Node_Component_Disk_Standard_Initialize
File: objects.nodes.components.disk.standard.F90
Used by:

Name: diskStarFormationTimescaleMinimum
Type: double 1
Default value: 1.0×10^{-3}
Description: The minimum timescale (in units of the halo dynamical time) on which star formation may occur in the disk.
Defined in: subroutine:Node_Component_Disk_Very_Simple_Initialize
File: objects.nodes.components.disk.very_simple.F90
Used by:

Name: diskStructureSolverRadius
Type: double 1
Default value: 1.0
Description: The radius (in units of the standard scale length) to use in solving for the size of the disk.
Defined in: subroutine:Node_Component_Disk_Standard_Initialize
File: objects.nodes.components.disk.standard.F90
Used by:

Name: diskVerySimpleAnalyticSolverPruneMassGas
Type: double 1
Default value: 0.0
Description: Gas mass below which the analytic solver will prune a galaxy.
Defined in: subroutine:Node_Component_Disk_Very_Simple_Initialize
File: objects.nodes.components.disk.very_simple.F90
Used by:

Name: diskVerySimpleAnalyticSolverPruneMassStars
Type: double 1
Default value: 0.0
Description: Stellar mass below which the analytic solver will prune a galaxy.
Defined in: subroutine:Node_Component_Disk_Very_Simple_Initialize
File: objects.nodes.components.disk.very_simple.F90
Used by:

Name: diskVerySimpleMassScaleAbsolute
Type: double 1
Default value: 100.0
Description: The absolute mass scale below which calculations in the very simple disk component are allowed to become inaccurate.
Defined in: subroutine:Node_Component_Disk_Very_Simple_Initialize
File: objects.nodes.components.disk.very_simple.F90
Used by:

Name: diskVerySimpleSurfaceDensityThreshold

Type: double 1
Default value: 0.0
Description: The threshold gas surface denisty above this star formation occurs [$M_\odot$/Mpc^2].
Defined in: subroutine:Node_Component_Disk_Very_Simple_Initialize
File: objects.nodes.components.disk.very_simple.F90
Used by:

Name: diskVerySimpleSurfaceDensityVelocityExponent
Type: double 1
Default value: 0.0
Description: The exponent of velocity in the threshold gas surface denisty above this star formation occurs.
Defined in: subroutine:Node_Component_Disk_Very_Simple_Initialize
File: objects.nodes.components.disk.very_simple.F90
Used by:

Name: diskVerySimpleTrackAbundances
Type: boolean 0..1
Default value: .false.
Description: Specifies whether or not to track abundances in the very simple disk component.
Defined in: subroutine:Node_Component_Disk_Very_Simple_Initialize
File: objects.nodes.components.disk.very_simple.F90
Used by:

Name: diskVerySimpleTrackLuminosities
Type: boolean 0..1
Default value: .false.
Description: Specifies whether or not to track stellar luminosities in the very simple disk component.
Defined in: subroutine:Node_Component_Disk_Very_Simple_Initialize
File: objects.nodes.components.disk.very_simple.F90
Used by:

Name: diskVerySimpleUseAnalyticSolver
Type: boolean 1
Default value: .false.
Description: If true, employ an analytic ODE solver when evolving satellites.
Defined in: subroutine:Node_Component_Disk_Very_Simple_Initialize
File: objects.nodes.components.disk.very_simple.F90
Used by:

Name: distanceMaximumSurvey
Type: float 0..1
Default value: 300.0×10^{-3}
Description: The maximum distance for the sample of classical Local Group galaxies.
Defined in: function:localGroupClassicalConstructorParameters
File: geometry.surveys.Local_Group_classical.F90
Used by:

Name: distributionComment

Type: string 0..1
Description: A descriptive comment for the distribution.
Defined in: function:volumeFunction1DConstructorParameters
File: galacticus.output.analyses.volume_function_1d.F90
Used by:

Name: distributionLabel
Type: string 0..1
Description: A label for the distribution.
Defined in: function:volumeFunction1DConstructorParameters
File: galacticus.output.analyses.volume_function_1d.F90
Used by:

Name: distributionNumber
Type: real 0..1
Description: The number (1-34) of the distribution to compute.
Defined in: function:galaxySizesSDSSConstructorParameters
File: galacticus.output.analyses.galaxy_sizes_SDSS.F90
Used by:

Name: distributionType
Type: string 1
Description: The name of the spin distribution to use.
Defined in: function:spinDistributionConstructorParameters
File: models.likelihoods.spin_distribution.F90
Used by:

Name: distributionUnits
Type: string 0..1
Description: A human-readable description of the units for the distribution.
Defined in: function:volumeFunction1DConstructorParameters
File: galacticus.output.analyses.volume_function_1d.F90
Used by:

Name: distributionUnitsInSI
Type: string 0..1
Description: A units for the distribution in the SI system.
Defined in: function:volumeFunction1DConstructorParameters
File: galacticus.output.analyses.volume_function_1d.F90
Used by:

Name: dummyEmulator
Type: boolean 1
Default value: .false.
Description: If true, then the emulator is constructed, and performance measured, but likelihoods are always simulated directly.
Defined in: function:gaussianRegressionConstructorParameters
File: models.likelihoods.gaussian_regression.F90
Used by:

Name: `dummyHostId`
Type: real 1
Description: If present, specifies the dummy host ID for self-hosting halos. Otherwise, self-hosting halos have `hostIndex == nodeIndex`.
Defined in: `function:mergerTreeFileBuilderConstructorParameters`
File: `tasks.merger_tree_file_builder.F90`
Used by:

Name: `dumpEmulatorFileRoot`
Type: string 1
Default value: var_str('')
Description: The name of a file to which emulator internal state will be dumped. (If empty, no dump occurs.)
Defined in: `function:gaussianRegressionConstructorParameters`
File: `models.likelihoods.gaussian_regression.F90`
Used by:

Name: `dumpTreeStructure`
Type: boolean 1
Default value: .false.
Description: Specifies whether merger tree structure should be dumped to a DOT file.
Defined in: `function:standardConstructorParameters`
File: `merger_trees.evolver.standard.F90`
Used by:

Name: `dumpTrees`
Type: boolean 1
Default value: .false.
Description: Specifies whether or not to dump merger trees as they are regridded.
Defined in: `function:regridTimesConstructorParameters`
File: `merger_trees.operators.regrid_times.F90`
Used by:

Name: `dustType`
Type: string 1
Description: The type of dust model to apply to the SED.
Defined in: `function:sedFitConstructorParameters`
File: `models.likelihoods.SED_fit.F90`
Used by:

Name: `dynamicalRateFraction`
Type: real 1
Default value: 2.0
Description: The fraction of the inverse dynamical time to use as the rate for infall of the cold mode component.
Defined in: `function:dynamicalTimeConstructorParameters`
File: `cooling.cold_mode.infall_rate.dynamical_time.F90`
Used by:

Name: `dynamicsStatisticsBarsFrequency`
Type: double 1
Default value: 0.1
Description: The frequency (in fractions of the host halo dynamical time) at which to record the bar dynamical status of satellite galaxies.
Defined in: `subroutine:Node_Component_Dynamics_Statistics_Bars_Initialize`
File: `objects.nodes.components.dynamics_statistics.bars.F90`
Used by:

Name: `edgeLengthsToTimes`
Type: boolean 1
Default value: .true.
Description: Specifies whether or not the lengths of edges in the graph should be scaled to time differences between nodes.
Defined in: `function:dumpToGraphVizConstructorParameters`
File: `merger_trees.operators.dump_to_GraphViz.F90`
Used by:

Name: `efficiency`
Type: float 1
Description: The fractional efficiency of two-body relaxation heating.
Defined in: `function:twoBodyRelaxationConstructorParameters`
File: `dark_matter_profiles.heating.two_body_relaxation.F90`
Used by:

Name: `efficiencyJet`
Type: real 0..1
Default value: 0.1
Description: The jet efficiency of the Eddington-limited accretion disk.
Defined in: `function:eddingtonLimitedConstructorParameters`
File: `accretion_disks.Eddington_limited.F90`
Used by:

Name: `efficiencyJetMaximum`
Type: real 1
Default value: 2.0
Description: The maximum efficiency allowed for ADAF-driven jets (in units of the accretion power).
Defined in: `function:adafConstructorParameters`
File: `accretion_disks.ADAF.F90`
Used by:

Name: `efficiencyRadiation`
Type: real 0..1
Default value: 0.1
Description: The radiative efficiency of the Eddington-limited accretion disk.
Defined in: `function:eddingtonLimitedConstructorParameters`
File: `accretion_disks.Eddington_limited.F90`
Used by:

Name: `efficiencyRadiationType`
Type: string 0..1
Default value: var_str('thinDisk')
Description: Specifies the specific energy of material at the inner edge of an ADAF. `pureADAF` makes the specific energy equal to 1 (i.e. all energy is advected with the flow); `ISCO` makes the specific energy equal to that for the innermost stable circular orbit.
Defined in: `function:adafConstructorParameters`
File: `accretion_disks.ADAF.F90`
Used by:

Name: `efficiencyRadiative`
Type: real 1
Default value: 2.75 [Covington et al., 2008]
Description: The coefficient, $C_{\rm rad}$ energy used in the Covington et al. [2008] merger remnant size algorithm.
Defined in: `function:covington2008ConstructorParameters`
File: `satellites.merging.remnant_sizes.Covington2008.F90`
Used by:

Name: `electronScatteringOpticalDepth`
Type: real 1
Description: The optical depth to reaionization in the instantReionization IGM state model.
Defined in: `function:instantReionizationIGMConstructorParameters`
File: `intergalactic_medium.state.instant_reionization.F90`
Used by:

Name: `element`
Type: string 0..1
Description: The atomic symbol for the element to use to define metallicity.
Defined in: `function:metallicityISMConstructorParameters`
File: `nodes.property_extractor.metallicity_ISM.F90`
Used by:

Name: `elementsToTrack`
Type: string 1..*
Description: The names of the elements to be tracked.
Defined in: `subroutine:Abundances_Initialize`
File: `objects.abundances.F90`
Used by:

Name: `emulateOutliers`
Type: boolean 1
Default value: .true.
Description: If true, then outlier chains are always emulated once the simulation is converged.
Defined in: `function:gaussianRegressionConstructorParameters`
File: `models.likelihoods.gaussian_regression.F90`
Used by:

Name: `emulatorRebuildCount`
Type: integer 1
Default value: 100
Description: The number of steps between rebuilds of the emulator.
Defined in: `function:gaussianRegressionConstructorParameters`
File: `models.likelihoods.gaussian_regression.F90`
Used by:

Name: `energyDistributionPointsPerDecade`
Type: real 0..1
Default value: 30.0
Description: The number of points per decade of radius to use when building the table of energy distribution function.
Defined in: `function:particulateConstructorParameters`
File: `merger_trees.operators.particulate.F90`
Used by:

Name: `energyEstimateParticleCountMaximum`
Type: real 1
Description: The maximum number of N-body particles used in estimating halo energies.
Defined in: `function:spinDistributionConstructorParameters`
File: `models.likelihoods.spin_distribution.F90`
Used by:

Name: `energyFixedAt`
Type: string 1
Default value: var_str('turnaround')
Description: Selects the epoch at which the energy of a spherical top hat perturbation in a dark energy cosmology should be "fixed" for the purposes of computing virial density contrasts. (See the discussion in Percival 2005; §8.)
Defined in: `function:sphericalCollapseMatterDEConstructorParameters`
File: `structure_formation.virial_density_contrast.spherical_collapse_matter_dark_energy.F90`
Used by:

Name: `energyOption`
Type: string 1
Default value: var_str('pureADAF')
Description: Specifies the specific energy of material at the inner edge of an ADAF. `pureADAF` makes the specific energy equal to 1 (i.e. all energy is advected with the flow); `ISCO` makes the specific energy equal to that for the innermost stable circular orbit.
Defined in: `function:adafConstructorParameters`
File: `accretion_disks.ADAF.F90`
Used by:

Name: `energyOrbital`
Type: real 1
Default value: 1.0
Description: The orbital energy in units of the characteristic orbital energy.
Defined in: `function:covington2008ConstructorParameters`

File: satellites.merging.remnant_sizes.Covington2008.F90
Used by:

Name: environmentAveraged
Type: boolean 1
Description: If true, the mass function will ve averaged over all environments.
Defined in: function:haloMassFunctionConstructorParameters
File: models.likelihoods.halo_mass_function.F90
Used by:

Name: epsilon
Type: real 1
Default value: 0.359 [Barkana et al., 2001, for the transfer function at $z = z_{\rm eq}$]
Description: The parameter ϵ appearing in the warm dark matter transfer function [Barkana et al., 2001].
Defined in: function:bode2001ConstructorParameters
File: structure_formation.transfer_function.Bode2001.F90
Used by:

Name: error
Type: real 1..*
Description: The errors on the magnitudes of the broad-band SED.
Defined in: function:sedFitConstructorParameters
File: models.likelihoods.SED_fit.F90
Used by:

Name: errorFractionalMaximum
Type: real 0..1
Description: Maximum allowed fractional error in halo mass.
Defined in: function:errorConvolvedConstructorParameters
File: structure_formation.halo_mass_function.error_convolved.F90
Used by:

Name: errorMaximum
Type: float 0..1
Description: The maximum error in the polynomial random error distribution class.
Defined in: function:randomErrorPolynomialConstructorParameters
File: galacticus.output.analyses.distribution_operator.random_error.polynomial.F90
Used by:

Name: errorMinimum
Type: float 0..1
Description: The minimum error in the polynomial random error distribution class.
Defined in: function:randomErrorPolynomialConstructorParameters
File: galacticus.output.analyses.distribution_operator.random_error.polynomial.F90
Used by:

Name: errorModel
Type: string 1

Description: The error model to use for the halo mass function.
Defined in: function:haloMassFunctionConstructorParameters
File: models.likelihoods.halo_mass_function.F90
Used by:

Name: errorWaitTime
Type: integer 1
Default value: 86400
Description: The time, in seconds, for which Galacticus should sleep after a fatal error when running under MPI.
Defined in: subroutine:Galacticus_Error_Wait_Set_From_Parameters
File: galacticus.error.wait.F90
Used by:

Name: essentialNodeID
Type: integer 1
Description: ID of the essential node to avoid pruning.
Defined in: function:pruneNonEssentialConstructorParameters
File: merger_trees.operators.prune_non_essential.F90
Used by:

Name: essentialNodeTime
Type: real 1
Description: Time of the essential node to avoid pruning.
Defined in: function:pruneNonEssentialConstructorParameters
File: merger_trees.operators.prune_non_essential.F90
Used by:

Name: eta
Type: real 1
Default value: 3.81 [Barkana et al., 2001, for the transfer function at $z = z_{\rm eq}$]
Description: The parameter ϵ appearing in the warm dark matter transfer function [Barkana et al., 2001].
Defined in: function:bode2001ConstructorParameters
File: structure_formation.transfer_function.Bode2001.F90
Used by:

Name: eta0
Type: real 1
Default value: 6.82
Description: The parameter η_0 appearing in the halo concentration algorithm of Diemer and Kravtsov [2014].
Defined in: function:diemerKravtsov2014ConstructorParameters
File: dark_matter_profiles.structure.concentration.Diemer-Kravtsov2014.F90
Used by:

Name: eta1
Type: real 1
Default value: 1.42

Description: The parameter η_1 appearing in the halo concentration algorithm of Diemer and Kravtsov [2014].
Defined in: function:diemerKravtsov2014ConstructorParameters
File: dark_matter_profiles.structure.concentration.Diemer-Kravtsov2014.F90
Used by:

Name: evolveForestsVerbosity
Type: integer 1
Default value: Galacticus_Verbosity_Level()
Description: The verbosity level to use while performing evolve forests tasks.
Defined in: function:galaxyPopulationConstructorParameters
File: models.likelihoods.galaxy_population.F90
Used by:

Name: evolveSingleForest
Type: boolean 1
Default value: .false.
Description: If true then each forest is processed sequentially, with multiple parallel threads (if available) working on the same forest. If false, multiple forests are processed simultaneously, with a single parallel thread (if available) working on each.
Defined in: function:evolveForestsConstructorParameters
File: tasks.evolve_forests.F90
Used by:

Name: evolveSingleForestMassMinimum
Type: integer 1
Default value: 0.0
Description: The minimum tree mass for which forests should be processed in parallel.
Defined in: function:evolveForestsConstructorParameters
File: tasks.evolve_forests.F90
Used by:

Name: evolveSingleForestSections
Type: integer 1
Default value: 100
Description: The number of timesteps into which forests should be split when processing single forests in parallel.
Defined in: function:evolveForestsConstructorParameters
File: tasks.evolve_forests.F90
Used by:

Name: exclusions
Type: string 1
Default value: [integer ::]
Description: List of parameter indices to exclude from the posterior likelihood calculation.
Defined in: function:posteriorAsPriorConstructorParameters
File: models.likelihoods.posterior_as_prior.F90
Used by:

Name: `expansionFactorEnd`
Type: real 1
Default value: 1.0
Description: Ending expansion factor to use when regridding merger trees.
Defined in: `function:regridTimesConstructorParameters`
File: `merger_trees.operators.regrid_times.F90`
Used by:

Name: `expansionFactorStart`
Type: real 1
Default value: 0.1
Description: Starting expansion factor to use when regridding merger trees.
Defined in: `function:regridTimesConstructorParameters`
File: `merger_trees.operators.regrid_times.F90`
Used by:

Name: `exponent`
Type: real 0..1
Default value: 0.8 Giocoli et al. [2008]
Description: The parameter α in the Giocoli et al. [2008] unevolved subhalo mass function fit.
Defined in: `function:giocoli2008ConstructorParameters`
File: `structure_formation.unevolved_subhalo_mass_function.Giocoli2008.F90`
Used by:

Name: `exponentAdjustFactor`
Type: real 1
Description: The factor by which to adjust the exponent.
Defined in: `function:adaptiveConstructorParameters`
File: `posterior_sampling.differential_proposal_size.temperature_exponent.adaptive.F90`
Used by:

Name: `exponentCutOff`
Type: real 1
Default value: 1.0
Description: The exponent appearing in the exponential term for cooling timescale in the velocity maximum scaling scaling cooling rate model.
Defined in: `function:velocityMaximumScalingConstructorParameters`
File: `cooling.cooling_rate.velocity_maximum_scaling.F90`
Used by:

Name: `exponentExpansionFactor`
Type: real 1
Default value: 0.0
Description: The exponent for expansion factor in the Baugh et al. [2005] prescription for star formation in galactic disks.
Defined in: `function:baugh2005ConstructorParameters`
File: `star_formation.timescales.disks.Baugh2005.F90`
Used by:

Name: `exponentGas`
Type: real 1
Default value: 1.0000 [Shi et al., 2011]
Description: The exponent of gas surface density in the extended Schmidt star formation law.
Defined in: `function:extendedSchmidtConstructorParameters`
File: `star_formation.rate_surface_density.disks.extended_Schmidt.F90`
Used by:

Name: `exponentInitial`
Type: real 1
Description: The initial exponent.
Defined in: `function:adaptiveConstructorParameters`
File: `posterior_sampling.differential_proposal_size.temperature_exponent.adaptive.F90`
Used by:

Name: `exponentMaximum`
Type: real 1
Description: The maximum allowed exponent.
Defined in: `function:adaptiveConstructorParameters`
File: `posterior_sampling.differential_proposal_size.temperature_exponent.adaptive.F90`
Used by:

Name: `exponentMinimum`
Type: real 1
Description: The minimum allowed exponent.
Defined in: `function:adaptiveConstructorParameters`
File: `posterior_sampling.differential_proposal_size.temperature_exponent.adaptive.F90`
Used by:

Name: `exponentRedshift`
Type: real 0..1
Default value: 0.0
Description: The exponent of redshift in the outflow rate in disks.
Defined in: `function:vlctyMxSclngConstructorParameters`
File: `star_formation.feedback.disks.velocity_maximum_scaling.F90`
Used by:

Name: `exponentStars`
Type: real 1
Default value: 0.4800 [Shi et al., 2011]
Description: The exponent of stellar surface density in the extended Schmidt star formation law.
Defined in: `function:extendedSchmidtConstructorParameters`
File: `star_formation.rate_surface_density.disks.extended_Schmidt.F90`
Used by:

Name: `exponentTruncated`
Type: real 1
Default value: 6.0
Description: The exponent of the $\Sigma_{\rm gas}/\Sigma_{\rm crit}$ term used in truncating the Kennicutt-Schmidt star for-

mation law.
Defined in: function:kennicuttSchmidtConstructorParameters
File: star_formation.rate_surface_density.disks.Kennicutt-Schmidt.F90
Used by:

Name: exponentValue
Type: real 1
Description: The exponent of temperature.
Defined in: function:fixedConstructorParameters
File: posterior_sampling.differential_proposal_size.temperature_exponent.fixed.F90
Used by:

Name: exponentVelocity
Type: real 0..1
Default value: -2.0d0
Description: The exponent of virial velocity in the outflow rate in disks.
Defined in: function:vlctyMxSclngConstructorParameters
File: star_formation.feedback.disks.velocity_maximum_scaling.F90
Used by:

Name: exponentVelocityVirial
Type: real 1
Default value: 0.0
Description: The exponent of virial velocity in the timescale for star formation in the halo scaling timescale model for disks.
Defined in: function:haloScalingConstructorParameters
File: star_formation.timescales.disks.halo_scaling.F90
Used by:

Name: exportFormat
Type: string 1
Default value: var_str('galacticus')
Description: The output format to use when exporting merger trees.
Defined in: function:exportConstructorParameters
File: merger_trees.operators.export.F90
Used by:

Name: extendedStatistics
Type: boolean 1
Default value: .true.
Description: Compute extended statistics (formation rate function, $N^{\rm th}$ most-massive progenitor mass functions, etc.)?
Defined in: function:conditionalMFConstructorParameters
File: merger_trees.operators.conditional_mass_function.F90
Used by:

Name: extentLower
Type: float 0..1
Default value: -huge(0.0d0)

Description: Lower extent for the normal distribution weight operator.
Defined in: function:normalConstructorParameters
File: galacticus.output.analyses.property_operator.normal.F90
Used by:

Name: extentUpper
Type: float 0..1
Default value: +huge(0.0d0)
Description: Upper extent for the normal distribution weight operator.
Defined in: function:normalConstructorParameters
File: galacticus.output.analyses.property_operator.normal.F90
Used by:

Name: extractApocenter
Type: boolean 1
Default value: .false.
Description: Specifies whether or not satellite orbital apocenter data (radius, velocity) should be extracted.
Defined in: function:satelliteOrbitalExtremaConstructorParameters
File: nodes.property_extractor.satellite_orbital_extrema.F90
Used by:

Name: extractPericenter
Type: boolean 1
Default value: .false.
Description: Specifies whether or not satellite orbital pericenter data (radius, velocity) should be extracted.
Defined in: function:satelliteOrbitalExtremaConstructorParameters
File: nodes.property_extractor.satellite_orbital_extrema.F90
Used by:

Name: f
Type: real 1
Default value: 1.0
Description: The parameter "f" which defines the scale of the Gaussian window.
Defined in: function:lagrangianChan2017ConstructorParameters
File: structure_formation.power_spectrum.variance.window_function.Lagrangian_Chan2017.F90
Used by:

Name: fSigma
Type: real 0..1
Default value: 0.4 [Obreschkow et al., 2009]
Description: The parameter, $\langle f_\sigma \rangle$, appearing in the model for the H_2/HI ratio in galaxies from Obreschkow et al. [2009].
Defined in: function:obreschkow2009ConstructorParameters
File: galacticus.output.analyses.molecular_ratio.Obreschkow2009.F90
Used by:

Name: factor

Type: real 0..1
Default value: 1.0
Description: The factor by which to rescale the excursion set barrier.
Defined in: function:remapScaleConstructorParameters
File: structure_formation.excursion_sets.barrier.remap.scale.F90
Used by:

Name: factorBoost
Type: float 1
Default value: 1.0
Description: The factor by which to boost satellite tidal fields in the sphericalSymmetry tidal field class.
Defined in: function:sphericalSymmetryConstructorParameters
File: satellites.tidal_fields.spherical_symmetry.F90
Used by:

Name: failedParametersFileName
Type: string 1
Default value: var_str('./failedParameters.xml')
Description: The filename to which parameters of failed models should be written.
Defined in: function:galaxyPopulationConstructorParameters
File: models.likelihoods.galaxy_population.F90
Used by:

Name: fast
Type: boolean 1
Default value: .true.
Description: Specifies whether or not to use simplifying assumptions to speed the hydrogen network calculation. If true, H^- is assumed to be at equilibrium abundance, H_2^+ reactions are ignored and other slow reactions are ignored (see Abel et al. 1997).
Defined in: function:hydrogenNetworkConstructorParameters
File: chemical.reaction_rates.hydrogen.F90
Used by:

Name: fatalMismatches
Type: boolean 1
Default value: .true.
Description: Specifies whether mismatches in cosmological parameter values between Galacticus and the merger tree file should be considered fatal.
Defined in: function:galacticusConstructorParameters
File: merger_trees.construct.read.importer.galacticus.F90
Used by:

Name: fatalNonTreeNode
Type: boolean 1
Default value: .true.
Description: Specifies whether nodes in snapshot files but not in the merger tree file should be considered fatal when importing from the "Sussing Merger Trees" format [Srisawat et al., 2013].
Defined in: function:sussingConstructorParameters

File: merger_trees.construct.read.importer.SussingMergerTrees.F90
Used by:

Name: fieldEnhancementOption
Type: string 1
Default value: var_str('exponential')
Description: Controls how the field enhancing shear is determined. exponential will cause the form $g = \exp(\omega t)$ [Benson and Babul, 2009] to be used, while linear will cause $g = 1 + \omega t$ to be used instead. The functional form of $\alpha(j)$ (if used) will be adjusted to achieve a sensible spin-up function in each case.
Defined in: function:adafConstructorParameters
File: accretion_disks.ADAF.F90
Used by:

Name: fileName
Type: string 0..1
Description: The name of the file from which to read a tabulated transfer function.
Defined in: function:fileConstructorParameters
File: structure_formation.transfer_function.file.F90
Used by:

Name: fileNames
Type: string 1..
Description: The name of the file(s) from which merger tree data should be read when using the [mergerTreeConstructMethod]=read tree construction method.
Defined in: function:readConstructorParameters
File: merger_trees.construct.read.F90
Used by:

Name: filter
Type: string 1..*
Description: The names of the filters in the broad-band SED.
Defined in: function:sedFitConstructorParameters
File: models.likelihoods.SED_fit.F90
Used by:

Name: filterName
Type: string 0..1
Description: The filter to select.
Defined in: function:lmnstyStllrChrltFll2000ConstructorParameters
File: nodes.property_extractor.luminosity_stellar.dust.CharlotFall2000.F90
Used by:

Name: filterThreshold
Type: float 0..1
Description: Threshold for the high-pass filter distribution operator.
Defined in: function:filterHighPassConstructorParameters
File: galacticus.output.analyses.property_operator.filter.high_pass.F90
Used by:

Name: `filterType`
Type: string 0..1
Description: The filter type (rest or observed) to select.
Defined in: function:lmnstyStllrChrltFll2000ConstructorParameters
File: nodes.property_extractor.luminosity_stellar.dust.CharlotFall2000.F90
Used by:

Name: `finalLikelihoodFullEvaluation`
Type: boolean 1
Default value: .true.
Description: If true the final likelihood is evaluated fully, and not treated as a "lock in" likelihood.
Defined in: function:independentLikelihoodsSequentialConstructorParameters
File: models.likelihoods.independent_likelihoods.sequential.F90
Used by:

Name: `fitType`
Type: string 1
Default value: var_str('nfwVirial')
Description: The type of halo definition for which the concentration-mass relation should be computed. Allowed values are `nfwVirial`, `nfwMean200`, `einastoMean200`, and `userDefined`.
Defined in: function:duttonMaccio2014ConstructorParameters
File: dark_matter_profiles.structure.concentration.Dutton-Maccio2014.F90
Used by:

Name: `forceZeroMetallicity`
Type: boolean 0..1
Default value: .false.
Description: Force the use of zero metallicity (or lowest metallicity available) for all stellar populations.
Defined in: function:fileConstructorParameters
File: stellar_populations.spectra.file.F90
Used by:

Name: `forestFile`
Type: string 1
Default value: var_str('none')
Description: Name of file containing data on number of halos in each forest.
Defined in: function:sussingASCIIConstructorParameters
File: merger_trees.construct.read.importer.SussingMergerTrees.ASCII.F90
Used by:

Name: `forestFirst`
Type: integer 1
Default value: 1
Description: Index of first forest to include.
Defined in: function:sussingASCIIConstructorParameters
File: merger_trees.construct.read.importer.SussingMergerTrees.ASCII.F90
Used by:

Name: `forestLast`

Type: integer 1
Default value: -1
Description: Index of last forest to include.
Defined in: function:sussingASCIIConstructorParameters
File: merger_trees.construct.read.importer.SussingMergerTrees.ASCII.F90
Used by:

Name: forestReverseSnapshotOrder
Type: integer 1
Default value: .false.
Description: If true, the order of forest snapshots will be reversed after being read. This may be necessary to cause them to match the order of snapshot files.
Defined in: function:sussingASCIIConstructorParameters
File: merger_trees.construct.read.importer.SussingMergerTrees.ASCII.F90
Used by:

Name: forestSizeMaximum
Type: integer 1
Default value: 0_c_size_t
Description: The maximum number of nodes allowed in a forest before it will be broken up into trees and processed individually. A value of 0 implies that forests should never be split.
Defined in: function:readConstructorParameters
File: merger_trees.construct.read.F90
Used by:

Name: formFactor
Type: real 1
Default value: 2.0
Description: The form factor appearing in the gravitational binding force (per unit area) in the ram pressure stripping model of Font et al. (2008; their eqn. 4).
Defined in: function:font2008ConstructorParameters
File: hot_halo.ram_pressure_stripping.Font2008.F90
Used by:

Name: formationRateTimeFraction
Type: real 1
Default value: 0.01
Description: The fraction of the current time over which to estimate the formation rate of halos when computing merger tree statistics.
Defined in: function:conditionalMFConstructorParameters
File: merger_trees.operators.conditional_mass_function.F90
Used by:

Name: formationRedshift
Type: real 1
Description: The formation redshift to use in Wechsler et al. [2002] halo mass accretion histories.
Defined in: function:wechsler2002ConstructorParameters
File: dark_matter_halos.mass_accretion_history.Wechsler2002.F90
Used by:

Name: formationRedshiftCompute
Type: boolean 1
Default value: .true.
Description: If true, compute formation redshift automatically for Wechsler et al. [2002] halo mass accretion histories.
Defined in: function:wechsler2002ConstructorParameters
File: dark_matter_halos.mass_accretion_history.Wechsler2002.F90
Used by:

Name: formationTimeMassFraction
Type: double 1
Default value: 0.5
Description: Fractional mass of primary progenitor used to define formation time.
Defined in: subroutine:Node_Component_Formation_Times_Mass_Fraction_Initialize
File: objects.nodes.components.formation_times.mass_fraction.F90
Used by:

Name: formationTimeRecent
Type: real 0..1
Description: Halos which "formed" more recently than this time in the past will be excluded from the analysis.
Defined in: function:concentrationDistributionCDMCOCOConstructorParameters
File: galacticus.output.analyses.concentration_distribution.CDM.COCO.F90
Used by:

Name: fraction
Type: real 0..1
Default value: 0.01
Description: The ratio of outflow rate to star formation rate in disks.
Defined in: function:vlctyMxSclngConstructorParameters
File: star_formation.feedback.disks.velocity_maximum_scaling.F90
Used by:

Name: fractionGasCriticalBurst
Type: real 1
Default value: 0.75
Description: The host gas fraction above which mergers are considered to trigger a burst.
Defined in: function:baugh2005ConstructorParameters
File: satellites.merging.mass_movements.Baugh2005.F90
Used by:

Name: fractionalErrorHighMass
Type: real 0..1
Description: Parameter σ_∞ appearing in model for random errors in the halo mass function.
Defined in: function:nbodyHaloMassErrorPowerLawParameters
File: statistics.Nbody.halos.mass_errors.power_law.F90
Used by:

Name: `frequencyStarFormation`
Type: real 1
Default value: 0.385 [Krumholz et al., 2009]
Description: The star formation frequency (in units of Gyr^{-1}) in the "Krumholz-McKee-Tumlinson" star formation timescale calculation.
Defined in: `function:krumholz2009ConstructorParameters`
File: `star_formation.rate_surface_density.disks.Krumholz2009.F90`
Used by:

Name: `functionCovarianceTarget`
Type: real 0..1
Description: The target function covariance for likelihood calculations.
Defined in: `function:volumeFunction1DConstructorParameters`
File: `galacticus.output.analyses.volume_function_1d.F90`
Used by:

Name: `functionValueTarget`
Type: real 0..1
Description: The target function for likelihood calculations.
Defined in: `function:volumeFunction1DConstructorParameters`
File: `galacticus.output.analyses.volume_function_1d.F90`
Used by:

Name: `galacticusOutputFileName`
Type: string 1
Default value: var_str('galacticus.hdf5')
Description: The name of the file to which Galacticus results will be written.
Defined in: `subroutine:Galacticus_Output_Open_File`
File: `galacticus.output.HDF5.open.F90`
Used by:

Name: `galacticusOutputScratchFileName`
Type: string 1
Default value: galacticusOutputFileName
Description: The name of the file to which Galacticus results will be written temporarily during runs.
Defined in: `subroutine:Galacticus_Output_Open_File`
File: `galacticus.output.HDF5.open.F90`
Used by:

Name: `galaxyCatalogFileName`
Type: string 1
Description: The file name to which the galaxy catalog should be output.
Defined in: `function:haloModelGenerateConstructorParameters`
File: `tasks.halo_model_generate.F90`
Used by:

Name: `gamma`
Type: real 1

Description: The parameter γ of the Voight function.
Defined in: function:voightConstructorParameters
File: statistics.distributions.Voight.F90
Used by:

Name: gamma1
Type: real 1
Default value: 0.38
Description: The parameter γ_1 appearing in the modified merger rate expression of Parkinson et al. [2008].
Defined in: function:parkinsonColeHellyConstructorParameters
File: merger_trees.branching_probability.Parkinson_Cole_Helly.F90
Used by:

Name: gamma2
Type: real 1
Default value: -0.01d0
Description: The parameter γ_2 appearing in the modified merger rate expression of Parkinson et al. [2008].
Defined in: function:parkinsonColeHellyConstructorParameters
File: merger_trees.branching_probability.Parkinson_Cole_Helly.F90
Used by:

Name: gammaAdjustFactor
Type: real 1
Description: The factor by which to adjust the proposal size, γ.
Defined in: function:adaptiveConstructorParameters
File: posterior_sampling.differential_proposal_size.adaptive.F90
Used by:

Name: gammaInitial
Type: real 1
Description: The initial proposal size, γ.
Defined in: function:adaptiveConstructorParameters
File: posterior_sampling.differential_proposal_size.adaptive.F90
Used by:

Name: gammaMaximum
Type: real 1
Description: The maximum allowed proposal size, γ.
Defined in: function:adaptiveConstructorParameters
File: posterior_sampling.differential_proposal_size.adaptive.F90
Used by:

Name: gammaMinimum
Type: real 1
Description: The minimum allowed proposal size, γ.
Defined in: function:adaptiveConstructorParameters
File: posterior_sampling.differential_proposal_size.adaptive.F90

Used by:

Name: gammaRatioHigh
Type: real 1
Default value: [+0.071d0,+0.030d0,-0.012d0]
Description: Values of the γ parameter of the Jiang et al. [2014] orbital velocity distribution for the three host halo mass ranges, and the 0.05–0.5 mass ratio.
Defined in: function:jiang2014ConstructorParameters
File: satellites.merging.virial_orbits.Jiang2014.F90
Used by:

Name: gammaRatioIntermediate
Type: real 1
Default value: [+0.098d0,+0.087d0,+0.050d0]
Description: Values of the γ parameter of the Jiang et al. [2014] orbital velocity distribution for the three host halo mass ranges, and the 0.005–0.05 mass ratio.
Defined in: function:jiang2014ConstructorParameters
File: satellites.merging.virial_orbits.Jiang2014.F90
Used by:

Name: gammaRatioLow
Type: real 1
Default value: [+0.109d0,+0.114d0,+0.110d0]
Description: Values of the γ parameter of the Jiang et al. [2014] orbital velocity distribution for the three host halo mass ranges, and the 0.0001–0.005 mass ratio.
Defined in: function:jiang2014ConstructorParameters
File: satellites.merging.virial_orbits.Jiang2014.F90
Used by:

Name: gradientMaximum
Type: real 1
Description: The maximum acceptable gradient of acceptance rate with log-temperature.
Defined in: function:adaptiveConstructorParameters
File: posterior_sampling.differential_proposal_size.temperature_exponent.adaptive.F90
Used by:

Name: gradientMinimum
Type: real 1
Description: The minimum acceptable gradient of acceptance rate with log-temperature.
Defined in: function:adaptiveConstructorParameters
File: posterior_sampling.differential_proposal_size.temperature_exponent.adaptive.F90
Used by:

Name: halfIntegral
Type: string 1
Default value: .false.
Description: Set to `true` if the projected correlation function is computed as $w_\mathrm{p}(r_\mathrm{p}) = \int_0^{+\pi_\mathrm{max}} \xi(r_\mathrm{p},\pi)\mathrm{d}\pi$, instead of the usual $w_\mathrm{p}(r_\mathrm{p}) = \int_{-\pi_\mathrm{max}}^{+\pi_\mathrm{max}} \xi(r_\mathrm{p},\pi)\mathrm{d}\pi$.
Defined in: function:haloModelProjectedCorrelationFunctionConstructorParameters

File: tasks.halo_model.projected_correlation_function.F90
Used by:

Name: haloCatalogFileName
Type: string 1
Description: The file name of the halo catalog to populate.
Defined in: function:haloModelGenerateConstructorParameters
File: tasks.halo_model_generate.F90
Used by:

Name: haloIdToParticleType
Type: boolean 0..1
Default value: .false.
Description: If true, the halo ID will be used to assign particles from each halo to a different Gadget particle type group.
Defined in: function:particulateConstructorParameters
File: merger_trees.operators.particulate.F90
Used by:

Name: haloMassMaximum
Type: real 1
Default value: 1.0×10^{15}
Description: The maximum mass at which to tabulate halo mass functions.
Defined in: function:haloMassFunctionConstructorParameters
File: tasks.halo_mass_function.F90
Used by:

Name: haloMassMinimum
Type: real 0..1
Default value: 0.0
Description: Minimum halo mass above which spin distribution should be averaged.
Defined in: function:haloSpinDistributionConstructorParameters
File: tasks.halo_spin_distribution.F90
Used by:

Name: haloMassesIncludeSubhalos
Type: boolean 1
Description: Specifies whether or not halo masses include the masses of their subhalos.
Defined in: function:mergerTreeFileBuilderConstructorParameters
File: tasks.merger_tree_file_builder.F90
Used by:

Name: haloModelWavenumberMaximum
Type: real 1
Default value: 1.0×10^{4}
Description: The maximum wavenumber (in Mpc^{-}1) at which to tabulate power spectra for the halo model.
Defined in: subroutine:Galacticus_Output_Halo_Model_Initialize
File: galacticus.output.merger_tree.halo_model.F90

Used by:

Name: haloModelWavenumberMinimum
Type: real 1
Default value: 1.0×10^{-3}
Description: The minimum wavenumber (in Mpc$^-$1) at which to tabulate power spectra for the halo model.
Defined in: subroutine:Galacticus_Output_Halo_Model_Initialize
File: galacticus.output.merger_tree.halo_model.F90
Used by:

Name: haloModelWavenumberPointsPerDecade
Type: integer 1
Default value: 10
Description: The number of points per decade in wavenumber at which to tabulate power spectra for the halo model.
Defined in: subroutine:Galacticus_Output_Halo_Model_Initialize
File: galacticus.output.merger_tree.halo_model.F90
Used by:

Name: haloReformationMassFactor
Type: double 1
Default value: 2.0
Description: Factor by which halo mass must have increased to trigger a new formation event.
Defined in: subroutine:Node_Component_Formation_Times_Cole2000_Initialize
File: objects.nodes.components.formation_times.Cole2000.F90
Used by:

Name: haloReformationOnPromotionOnly
Type: boolean 1
Default value: .false.
Description: Specifies whether halo reformation should occur only at node promotion events, or at the precise time that the halo mass has increased sufficiently in mass.
Defined in: subroutine:Node_Component_Formation_Times_Cole2000_Initialize
File: objects.nodes.components.formation_times.Cole2000.F90
Used by:

Name: hdf5CacheElementsCount
Type: boolean 1
Default value: 521_size_t
Description: Number of elements in the raw data chunk cache.
Defined in: subroutine:Galacticus_Output_Open_File
File: galacticus.output.HDF5.open.F90
Used by:

Name: hdf5CacheSizeBytes
Type: boolean 1
Default value: 1048576_size_t
Description: Size of the raw data chunk cache in bytes.

Defined in: subroutine:Galacticus_Output_Open_File
File: galacticus.output.HDF5.open.F90
Used by:

Name: hdf5ChunkSize
Type: integer 1
Default value: 1024_hsize_t
Description: The chunk size used for outputting HDF5 datasets.
Defined in: subroutine:Galacticus_Output_Open_File
File: galacticus.output.HDF5.open.F90
Used by:

Name: hdf5CompressionLevel
Type: integer 1
Default value: -1
Description: The compression level used for outputting HDF5 datasets.
Defined in: subroutine:Galacticus_Output_Open_File
File: galacticus.output.HDF5.open.F90
Used by:

Name: hdf5SieveBufferSize
Type: integer 1
Default value: 65536
Description: The size of the sieve buffer used by the HDF5 library to speed reading/writing of partial datasets.
Defined in: subroutine:Galacticus_Output_Open_File
File: galacticus.output.HDF5.open.F90
Used by:

Name: hdf5UseLatestFormat
Type: boolean 1
Default value: .false.
Description: Specifies whether to use the latest HDF5 file format.
Defined in: subroutine:Galacticus_Output_Open_File
File: galacticus.output.HDF5.open.F90
Used by:

Name: heightToRadialScaleDisk
Type: real 1
Default value: 0.137 [Kregel et al., 2002]
Description: The ratio of scale height to scale radius for disks in the "Blitz-Rosolowsky2006" star formation timescale calculation.
Defined in: function:blitz2006ConstructorParameters
File: star_formation.rate_surface_density.disks.Blitz2006.F90
Used by:

Name: hierarchyDepth
Type: integer 1
Default value: 1

Description: The depth in the substructure hierarchy at which to prune a tree.
Defined in: function:pruneHierarchyConstructorParameters
File: merger_trees.operators.prune_hierarchy.F90
Used by:

Name: hierarchyLevelResetFactor
Type: double 1
Default value: 1.0×10^{100}
Description: The factor by which a node's mass must increase before the previous maximum hierarchy level is forgotten.
Defined in: subroutine:Node_Component_Merging_Statistics_Standard_Initialize
File: objects.nodes.components.merging_statistics.standard.F90
Used by:

Name: historyCount
Type: integer 1
Default value: 30
Description: The number of steps (spaced logarithmically in cosmic time) at which to tabulate the volume averaged history of galaxies.
Defined in: function:historyConstructorParameters
File: merger_trees.evolve.timesteps.history.F90
Used by:

Name: hotHaloAngularMomentumAlwaysGrows
Type: boolean 1
Default value: .false.
Description: Specifies whether or not negative rates of accretion of angular momentum into the hot halo will be treated as positive for the purposes of computing the hot halo angular momentum.
Defined in: subroutine:Node_Component_Hot_Halo_Standard_Initialize
File: objects.nodes.components.hot_halo.standard.F90
Used by:

Name: hotHaloAngularMomentumLossFraction
Type: double 1
Default value: 0.3
Description: Specifies the fraction of angular momentum that is lost from cooling/infalling gas.
Defined in: subroutine:Node_Component_Hot_Halo_Standard_Initialize
File: objects.nodes.components.hot_halo.standard.F90
Used by:

Name: hotHaloCoolingFromNode
Type: integer 1
Default value: var_str('currentNode')
Description: Specifies whether the angular momentum of cooling gas should be computed from the "current node" or the "formation node".
Defined in: subroutine:Node_Component_Hot_Halo_Standard_Initialize
File: objects.nodes.components.hot_halo.standard.F90
Used by:

Name: `hotHaloExcessHeatDrivesOutflow`
Type: integer 1
Default value: .true.
Description: Specifies whether heating of the halo in excess of its cooling rate will drive an outflow from the halo.
Defined in: subroutine:Node_Component_Hot_Halo_Standard_Initialize
File: objects.nodes.components.hot_halo.standard.F90
Used by:

Name: `hotHaloExpulsionRateMaximum`
Type: double 1
Default value: 1.0
Description: Specifies the maximum rate at which mass can be expelled from the hot halo in units of the inverse halo dynamical time.
Defined in: subroutine:Node_Component_Hot_Halo_Standard_Initialize
File: objects.nodes.components.hot_halo.standard.F90
Used by:

Name: `hotHaloNodeMergerLimitBaryonFraction`
Type: boolean 1
Default value: .false.
Description: Controls whether the hot gas content of nodes should be limited to not exceed the universal baryon fraction at node merger events. If set to `true`, hot gas (and angular momentum, abundances, and chemicals proportionally) will be removed from the merged halo to the unaccreted gas reservoir to limit the baryonic mass to the universal baryon fraction where possible.
Defined in: subroutine:Node_Component_Hot_Halo_Standard_Initialize
File: objects.nodes.components.hot_halo.standard.F90
Used by:

Name: `hotHaloOutflowReturnOnFormation`
Type: boolean 1
Default value: .false.
Description: Specifies whether or not outflowed gas should be returned to the hot reservoir on halo formation events.
Defined in: subroutine:Node_Component_Hot_Halo_Standard_Initialize
File: objects.nodes.components.hot_halo.standard.F90
Used by:

Name: `hotHaloOutflowStrippingEfficiency`
Type: double 1
Default value: 0.1
Description: Specifies the efficiency with which outflowing gas is stripped from the hot halo, following the prescription of Font et al. (2008; i.e. this is the parameter $\epsilon_{\rm strip}$ in their eqn. 6).
Defined in: subroutine:Node_Component_Hot_Halo_Standard_Initialize
File: objects.nodes.components.hot_halo.standard.F90
Used by:

Name: `hotHaloOutflowToColdMode`
Type: boolean 1

Default value: .false.
Description: Specifies whether or not outflows from galaxies are returned to the cold or hot modes in the hot halo.
Defined in: subroutine:Node_Component_Hot_Halo_Cold_Mode_Initialize
File: objects.nodes.components.hot_halo.cold_mode.F90
Used by:

Name: hotHaloTrackStrippedGas
Type: boolean 1
Default value: .true.
Description: Specifies whether or not gas stripped from the hot halo should be tracked.
Defined in: subroutine:Node_Component_Hot_Halo_Standard_Initialize
File: objects.nodes.components.hot_halo.standard.F90
Used by:

Name: hotHaloVerySimpleDelayedMassScaleRelative
Type: double 1
Default value: 1.0×10^{-2}
Description: The mass scale, relative to the total mass of the node, below which calculations in the delayed very simple hot halo component are allowed to become inaccurate.
Defined in: subroutine:Node_Component_Hot_Halo_VS_Delayed_Initialize
File: objects.nodes.components.hot_halo.very_simple_delayed.F90
Used by:

Name: hubbleExponent
Type: integer 1
Description: The exponent of the "little-h" Hubble parameter needed to convert the velocities to little-h-free units.
Defined in: function:mergerTreeFileBuilderConstructorParameters
File: tasks.merger_tree_file_builder.F90
Used by:

Name: hypergeometricTabulate
Type: boolean 1
Default value: .true.
Description: Specifies whether hypergeometric factors should be precomputed and tabulated in modified Press-Schechter tree branching functions.
Defined in: function:parkinsonColeHellyConstructorParameters
File: merger_trees.branching_probability.Parkinson_Cole_Helly.F90
Used by:

Name: idMultiplier
Type: real 0..1
Default value: 0_kind_int8
Description: If this parameter is greater than zero, particle IDs begin at nodeIndex[idMultiplier] for each node. The multiplier should be chosen to be large enough that duplicate IDs can not occur.
Defined in: function:particulateConstructorParameters
File: merger_trees.operators.particulate.F90
Used by:

Name: `ignoreChainNumberAdvice`
Type: logical 1
Default value: .false.
Description: If true, ignore warnings and errors about not being able to span the full parameter space with the number of chains used.
Defined in: `function:differentialEvolutionConstructorParameters`
File: `posterior_sampling.simulation.differential_evolution.F90`
Used by:

Name: `includeAngularCoordinates`
Type: boolean 0..1
Default value: .false.
Description: If true output angular coordinates in the lightcone.
Defined in: `function:lightconeConstructorParameters`
File: `nodes.property_extractor.lightcone.F90`
Used by:

Name: `includeBaryonGravity`
Type: boolean 1
Default value: .true.
Description: Specifies whether or not gravity from baryons is included when solving for sizes of galactic components in equilibriumally contracted dark matter halos.
Defined in: `function:equilibriumConstructorParameters`
File: `galactic_structure.radius_solver.equilibrium.F90`
Used by:

Name: `includeHalo`
Type: boolean 1
Default value: .true.
Description: Specifies whether or not to include the halo contribution to mass function covariance matrices.
Defined in: `function:massFunctionCovarianceConstructorParameters`
File: `tasks.mass_function_covariance.F90`
Used by:

Name: `includeLSS`
Type: boolean 1
Default value: .true.
Description: Specifies whether or not to include the large-scale structure contribution to mass function covariance matrices.
Defined in: `function:massFunctionCovarianceConstructorParameters`
File: `tasks.mass_function_covariance.F90`
Used by:

Name: `includeNitrogenII`
Type: real 0..1
Default value: .false.
Description: If true, include contimination by the [NII] (6548Å+ 6584Å) doublet.

Defined in: function:luminosityFunctionHalphaConstructorParameters
File: galacticus.output.analyses.luminosity_function.Halpha.F90
Used by:

Name: includeNonLinear
Type: integer 1
Default value: .false.
Description: If true the nonlinear power spectrum is also computed and output.
Defined in: function:powerSpectraConstructorParameters
File: tasks.power_spectrum.F90
Used by:

Name: includeObservedRedshift
Type: boolean 0..1
Default value: .false.
Description: If true output the observed redshfit (i.e. including the effects of pecular velocities).
Defined in: function:lightconeConstructorParameters
File: nodes.property_extractor.lightcone.F90
Used by:

Name: includePoisson
Type: boolean 1
Default value: .true.
Description: Specifies whether or not to include the Poisson contribution to mass function covariance matrices.
Defined in: function:massFunctionCovarianceConstructorParameters
File: tasks.mass_function_covariance.F90
Used by:

Name: includeRadii
Type: boolean 1
Default value: .false.
Description: Specifies whether or not the radii at which velocity dispersion data are output should also be included in the output file.
Defined in: function:velocityDispersionConstructorParameters
File: nodes.property_extractor.velocity_dispersion.F90
Used by:

Name: includeSpin
Type: boolean 1
Default value: .false.
Description: If true, include the spin of the halo in the output.
Defined in: function:massAccretionHistoryConstructorParameters
File: merger_trees.operators.mass_accretion_history.F90
Used by:

Name: includeSpinVector
Type: boolean 1
Default value: .false.

Description: If true, include the spin vector of the halo in the output.
Defined in: function:massAccretionHistoryConstructorParameters
File: merger_trees.operators.mass_accretion_history.F90
Used by:

Name: includesHubbleFlow
Type: boolean 1
Description: Specifies whether or not Hubble flow is included in velocities.
Defined in: function:mergerTreeFileBuilderConstructorParameters
File: tasks.merger_tree_file_builder.F90
Used by:

Name: index
Type: real 0..1
Default value: 0.9649 (Planck Collaboration et al. 2018; TT,TE,EE+lowE+lensing)
Description: The index of the power-law primordial power spectrum.
Defined in: function:powerLawConstructorParameters
File: structure_formation.power_spectrum.primordial.power_law.F90
Used by:

Name: inertiaWeight
Type: real 1
Default value: 0.72
Description: Inertia parameter.
Defined in: function:particleSwarmConstructorParameters
File: posterior_sampling.simulation.particle_swarm.F90
Used by:

Name: initialMassForSupernovaeTypeII
Type: real 1
Default value: 8.0
Description: The minimum mass that a star must have in order that is result in a Type II supernova.
Defined in: function:standardConstructorParameters
File: stellar_astrophysics.feedback.standard.F90
Used by:

Name: initializeNodeClassHierarchy
Type: boolean 1
Default value: .true.
Description: If true then initialize the node class hierarchy in the posterior sampling class. This should be set to false if the likelihood function will instead perform this action.
Defined in: function:posteriorSampleConstructorParameters
File: tasks.posterior_sample.F90
Used by:

Name: inputFileName
Type: string 1
Description: The name of the file from which to read merger tree data.
Defined in: function:mergerTreeFileBuilderConstructorParameters

File: tasks.merger_tree_file_builder.F90
Used by:

Name: instantaneousRecyclingApproximation
Type: boolean 1
Default value: .false.
Description: If true, then use an instantaneous recycling approximation when computing recycling, yield, and energy input rates.
Defined in: function:standardConstructorParameters
File: stellar_populations.standard.F90
Used by:

Name: integralConstraint
Type: real 0..1
Description: The integral constraint for these correlation functions.
Defined in: function:correlationFunctionConstructorParameters
File: galacticus.output.analyses.correlation_function.F90
Used by:

Name: interactionRoot
Type: integer 1
Default value: var_str('none')
Description: Root file name for interaction files, or '**none**' if interaction is not required.
Defined in: function:particleSwarmConstructorParameters
File: posterior_sampling.simulation.particle_swarm.F90
Used by:

Name: inverseSigma0
Type: real 1
Default value: 1.047 Prada et al. [2011]
Description: The parameter σ_0^{-1} appearing in the halo concentration algorithm of Prada et al. [2011].
Defined in: function:prada2011ConstructorParameters
File: dark_matter_profiles.structure.concentration.Prada2011.F90
Used by:

Name: inverseSigma1
Type: real 1
Default value: 1.646 Prada et al. [2011]
Description: The parameter σ_1^{-1} appearing in the halo concentration algorithm of Prada et al. [2011].
Defined in: function:prada2011ConstructorParameters
File: dark_matter_profiles.structure.concentration.Prada2011.F90
Used by:

Name: kappa
Type: real 1
Default value: 0.69
Description: The parameter κ appearing in the halo concentration algorithm of Diemer and Kravtsov [2014].
Defined in: function:diemerKravtsov2014ConstructorParameters

File: dark_matter_profiles.structure.concentration.Diemer-Kravtsov2014.F90
Used by:

Name: kernelSoftening
Type: string 0..1
Default value: var_str('plummer')
Description: Selects the softening kernel to use. Allowed options are "plummer", and "gadget".
Defined in: function:particulateConstructorParameters
File: merger_trees.operators.particulate.F90
Used by:

Name: label
Type: string 0..1
Description: A label for the analysis.
Defined in: function:volumeFunction1DConstructorParameters
File: galacticus.output.analyses.volume_function_1d.F90
Used by:

Name: lambda0
Type: real 1
Default value: 0.04326 [Bett et al., 2007]
Description: The parameter λ_0 in the halo spin distribution of Bett et al. [2007].
Defined in: function:bett2007ConstructorParameters
File: dark_matter_halos.spins.distributions.Bett2007.F90
Used by:

Name: lengthBox
Type: real 0..1
Default value: 0.0
Description: The length of the periodic box.
Defined in: function:environmentalOverdensityConstructorParameters
File: nBody.operator.environmental_overdensity.F90
Used by:

Name: lengthHubbleExponent
Type: real 0..1
Description: The exponent of the "little-h" parameter used in the definition of the box length.
Defined in: function:squareConstructorParameters
File: geometry.lightcones.square.F90
Used by:

Name: lengthReplication
Type: real 0..1
Description: The length of the simulation box being used to construct the lightcone.
Defined in: function:squareConstructorParameters
File: geometry.lightcones.square.F90
Used by:

Name: lengthSoftening

Type: boolean 0..1
Default value: 0.0
Description: The Plummer-equivalent softening length. That is, the parameter ϵ in the softening gravitational potential $\phi(r) = -\mathrm{G}m/\sqrt{r^2+\epsilon^2}$. If set to zero, softening is ignored when constructing the particle representation of the halo. For non-zero values softening is accounted for when constructing the velocity distribution following the procedure of Barnes [2012].
Defined in: function:particulateConstructorParameters
File: merger_trees.operators.particulate.F90
Used by:

Name: lengthUnitsInSI
Type: real 0..1
Description: The units of the box length in the SI system.
Defined in: function:squareConstructorParameters
File: geometry.lightcones.square.F90
Used by:

Name: likelihoodBin
Type: integer 0..1
Default value: 0_c_size_t
Description: If > 0 then use only the mass bin given by this value in the likelihood calculation.
Defined in: function:stellarVsHaloMassRelationLeauthaud2012ConstructorParameters
File: galacticus.output.analyses.stellar_vs_halo_mass_relation.COSMOS_Leauthaud2012.F90
Used by:

Name: likelihoodNormalize
Type: boolean 0..1
Default value: .true.
Description: If true then normalize the likelihood to make it a probability density.
Defined in: function:volumeFunction1DConstructorParameters
File: galacticus.output.analyses.volume_function_1d.F90
Used by:

Name: likelihoodThreshold
Type: real 1
Default value: 0.0
Description: The threshold log-likelihood above which convergence is declared.
Defined in: function:likelihoodThresholdConstructorParameters
File: posterior_sampling.convergence.likelihood_threshold.F90
Used by:

Name: limit
Type: string 1
Description: Limiting surface brightness for mass function incompleteness calculations.
Defined in: function:surfaceBrightnessConstructorParameters
File: statistics.mass_function.incompleteness.surface_brightness.F90
Used by:

Name: limitLoadAverage

Type: boolean 1
Default value: .false.
Description: Specifies whether or not to limit the load average
Defined in: function:evolveForestsConstructorParameters
File: tasks.evolve_forests.F90
Used by:

Name: limitLower
Type: real 1
Description: The lower limit of the uniform distribution.
Defined in: function:uniformConstructorParameters
File: statistics.distributions.uniform.F90
Used by:

Name: limitUpper
Type: real 1
Description: The upper limit of the uniform distribution.
Defined in: function:uniformConstructorParameters
File: statistics.distributions.uniform.F90
Used by:

Name: lineNames
Type: string 0..1
Description: The emission lines to extract.
Defined in: function:lmnstyEmssnLineConstructorParameters
File: nodes.property_extractor.luminosity_emission_line.F90
Used by:

Name: lineOfSightDepth
Type: real 1
Description: The line of sight depth over which to integrate.
Defined in: function:projectedCorrelationFunctionConstructorParameters
File: models.likelihoods.projected_correlation_function.F90
Used by:

Name: linkingLength
Type: real 0..1
Description: The linking length (in physical Mpc) used in the friends-of-friends algorithm.
Defined in: function:fofBiasConstructorParameters
File: structure_formation.halo_mass_function.friends_of_friends_bias.F90
Used by:

Name: linkingLengthIsComoving
Type: boolean 0..1
Description: Specifies whether or not the given linking length is in comoving units.
Defined in: function:fofBiasConstructorParameters
File: structure_formation.halo_mass_function.friends_of_friends_bias.F90
Used by:

Name: `loadAverageMaximum`
Type: boolean 1
Default value: var_str('processorCount')
Description: The maximum load average for which new trees will be processed.
Defined in: `function:evolveForestsConstructorParameters`
File: `tasks.evolve_forests.F90`
Used by:

Name: `loadBalance`
Type: logical 1
Default value: .true.
Description: If true, attempt to balance the workload across different compute nodes.
Defined in: `function:differentialEvolutionConstructorParameters`
File: `posterior_sampling.simulation.differential_evolution.F90`
Used by:

Name: `lockFileGlobally`
Type: boolean 1
Default value: .false.
Description: If true, attempt to lock the CAMB transfer function file before accessing. Otherwise, no locking is attempted.
Defined in: `function:cambConstructorParameters`
File: `structure_formation.transfer_function.CAMB.F90`
Used by:

Name: `log10M1`
Type: real 1
Default value: 12.520 (Leauthaud et al. 2011; z_1 sample using their `SIG_MOD1` method)
Description: The parameter $\log_{10} M_1$ from the fitting functions of Behroozi et al. [2010].
Defined in: `function:behroozi2010ConstructorParameters`
File: `halo_model.conditional_mass_function.Behroozi2010.F90`
Used by:

Name: `log10Mstar0`
Type: real 1
Default value: 10.916 (Leauthaud et al. 2011; z_1 sample using their `SIG_MOD1` method)
Description: The parameter $\log_{10} M_{\star,0}$ from the fitting functions of Behroozi et al. [2010].
Defined in: `function:behroozi2010ConstructorParameters`
File: `halo_model.conditional_mass_function.Behroozi2010.F90`
Used by:

Name: `logFileName`
Type: real 1
Description: The name of the file to which convergence state should be logged.
Defined in: `function:gelmanRubinConstructorParameters`
File: `posterior_sampling.convergence.Gelman-Rubin.F90`
Used by:

Name: `logFilePreviousRoot`

Type: integer 1
Default value: var_str('none')
Description: Root file name for log files from which to resume.
Defined in: function:particleSwarmConstructorParameters
File: posterior_sampling.simulation.particle_swarm.F90
Used by:

Name: logFileRoot
Type: integer 1
Description: Root file name for log files.
Defined in: function:particleSwarmConstructorParameters
File: posterior_sampling.simulation.particle_swarm.F90
Used by:

Name: logFlushCount
Type: integer 1
Default value: 10
Description: The number of steps between flushing the log file.
Defined in: function:particleSwarmConstructorParameters
File: posterior_sampling.simulation.particle_swarm.F90
Used by:

Name: logLikelihoodBuffer
Type: real 1
Default value: 10.0
Description: The buffer size in log-likelihood to use when deciding whether to emulate the likelihood.
Defined in: function:gaussianRegressionConstructorParameters
File: models.likelihoods.gaussian_regression.F90
Used by:

Name: logLikelihoodErrorTolerance
Type: real 1
Default value: 0.1
Description: The tolerance on the likelihood error to accept when deciding whether to emulate the likelihood.
Defined in: function:gaussianRegressionConstructorParameters
File: models.likelihoods.gaussian_regression.F90
Used by:

Name: logM0
Type: real 0..1
Default value: 10.0
Description: The parameter $\log_{10} M_0$ (with M_0 in units of $M_\odot$) appearing in the star formation rate threshold expression for the star formation rate galactic filter class.
Defined in: function:starFormationRateConstructorParameters
File: galactic.filters.star_formation_rate.F90
Used by:

Name: logSFR0

Type: real 0..1
Default value: 9.0
Description: The parameter α_0 appearing in the star formation rate threshold expression for the star formation rate galactic filter class.
Defined in: function:starFormationRateConstructorParameters
File: galactic.filters.star_formation_rate.F90
Used by:

Name: logSFR1
Type: real 0..1
Default value: 0.0
Description: The parameter α_1 appearing in the star formation rate threshold expression for the star formation rate galactic filter class.
Defined in: function:starFormationRateConstructorParameters
File: galactic.filters.star_formation_rate.F90
Used by:

Name: logarithmCoulomb
Type: real 1
Default value: 2.0
Description: The Coulomb logarithm, $\ln \Lambda$, appearing in the Chandrasekhar [1943] formulation of the acceleration due to dynamical friction.
Defined in: function:chandrasekhar1943ConstructorParameters
File: satellites.dynamical_friction.acceleration.Chandrasekhar1943.F90
Used by:

Name: luminosities
Type: float 0..1
Description: The luminosities corresponding to bin centers.
Defined in: function:luminosityFunctionHalphaConstructorParameters
File: galacticus.output.analyses.luminosity_function.Halpha.F90
Used by:

Name: luminosityBandRedshift
Type: real 0..*
Description: If present, force filters to be shifted to this redshift rather than that specified by [luminosityRedshift]. Allows sampling of the SED at wavelengths corresponding to other redshifts.
Defined in: subroutine:Stellar_Luminosities_Initialize
File: objects.stellar_luminosities.F90
Used by:

Name: luminosityFilter
Type: string 0..*
Description: The filter name for each stellar luminosity to be computed.
Defined in: subroutine:Stellar_Luminosities_Initialize
File: objects.stellar_luminosities.F90
Used by:

Name: luminosityOutputOption

Type: integer 1
Default value: var_str('present')
Description: Selects which luminosities will be output at each output time:

all Output all luminosities;

future Output only those luminosities computed for the present output or future times;

present Output only those luminosities computed for the present output time.

Defined in: subroutine:Stellar_Luminosities_Initialize
File: objects.stellar_luminosities.F90
Used by:

Name: luminosityPostprocessSet
Type: string 0..*
Description: The name of the set of postprocessing algorithms to apply to this filter.
Defined in: subroutine:Stellar_Luminosities_Initialize
File: objects.stellar_luminosities.F90
Used by:

Name: luminosityRedshift
Type: real 0..*
Description: The redshift for which to compute each specified stellar luminosity.
Defined in: subroutine:Stellar_Luminosities_Initialize
File: objects.stellar_luminosities.F90
Used by:

Name: luminosityType
Type: string 0..*
Description: The luminosity type for each stellar luminosity to be computed:

rest Compute luminosity in the galaxy rest frame;

observed Compute luminosity in the observer frame[1].

Defined in: subroutine:Stellar_Luminosities_Initialize
File: objects.stellar_luminosities.F90
Used by:

Name: magnitude
Type: real 1..*
Description: The magnitudes of the broad-band SED.
Defined in: function:sedFitConstructorParameters
File: models.likelihoods.SED_fit.F90
Used by:

Name: magnitudesAbsolute
Type: float 0..1

[1] The luminosity computed in this way is that in the galaxy rest frame using a filter blueshifted to the galaxy's redshift. This means that to compute an apparent magnitude you must add not only the distance modulus, but a factor of $-2.5\log_{10}(1+z)$ to account for compression of photon frequencies.

Description: The absolute magnitudes corresponding to bin centers.
Defined in: function:luminosityFunctionConstructorParameters
File: galacticus.output.analyses.luminosity_function.F90
Used by:

Name: mass
Type: real 1..*
Default value: [0.1d0,125.0d0]
Description: The mass points used to define a piecewise power-law initial mass function.
Defined in: function:piecewisePowerLawConstructorParameters
File: stellar_populations.initial_mass_functions.piecewise_power_law.F90
Used by:

Name: massBinCenters
Type: real 1
Description: Logarithmic mass bins centers for conditional mass function calculations.
Defined in: function:conditionalMassFunctionConstructorParameters
File: tasks.conditional_mass_function.F90
Used by:

Name: massBinsPerDecade
Type: integer 1
Default value: 10
Description: The number of bins per decade of non-primary progenitor mass.
Defined in: function:profilerConstructorParameters
File: merger_trees.operators.profiler.F90
Used by:

Name: massCharacteristic
Type: real 1
Default value: 0.08
Description: Characteristic mass of the lognormal part of the Chabrier [2001] initial mass function (IMF).
Defined in: function:chabrier2001ConstructorParameters
File: stellar_populations.initial_mass_functions.Chabrier2001.F90
Used by:

Name: massCutOff
Type: double precision 0..
Default value: 1.0×10^{10}
Description: For the augment operator a description of resolution limit for new trees.
Defined in: function:augmentConstructorParameters
File: merger_trees.operators.augment.F90
Used by:

Name: massCutOffAttemptsMaximum
Type: integer 0..
Default value: 50
Description: The number of trees with nodes above the mass resolution to allow before adjusting the

mass cut-off tolerance.
Defined in: function:augmentConstructorParameters
File: merger_trees.operators.augment.F90
Used by:

Name: massCutOffScaleFactor
Type: real 0..
Default value: 0.05
Description: The amount by which to increase the mass cut-off scale tolerance after exhausting tree build attempts.
Defined in: function:augmentConstructorParameters
File: merger_trees.operators.augment.F90
Used by:

Name: massElement
Type: float 0..1
Description: The atomic mass of the element used to define metallicity.
Defined in: function:metallicity12LogNHConstructorParameters
File: galacticus.output.analyses.property_operator.metallicity.F90
Used by:

Name: massFlowStatisticsResetOnOutput
Type: double 1
Default value: .true.
Description: Specifies whether or not mass flow statistics should be reset to zero at each output.
Defined in: subroutine:Node_Component_Mass_Flow_Statistics_Standard_Initialize
File: objects.nodes.components.mass_flow_statistics.standard.F90
Used by:

Name: massFractionFormation
Type: real 1
Default value: 0.05
Description: The fraction of a halo's mass assembled at "formation" in the halo concentration algorithm of Schneider [2015].
Defined in: function:schneider2015ConstructorParameters
File: dark_matter_profiles.structure.concentration.Schneider2015.F90
Used by:

Name: massFunctionFileName
Type: string 1
Description: The name of the file to which the covariance matrix should be written.
Defined in: function:massFunctionCovarianceConstructorParameters
File: tasks.mass_function_covariance.F90
Used by:

Name: massFunctionType
Type: string 1
Description: The type of mass function () model to use.
Defined in: function:haloMassFunctionConstructorParameters

File: models.likelihoods.halo_mass_function.F90
Used by:

Name: massHalo
Type: string 1
Default value: var_str('all')
Description: The halo mass for which to compute the conditional mass function. A value of "all" will cause the conditional mass function to be integrated over the halo mass function, giving the mass function.
Defined in: function:conditionalMassFunctionConstructorParameters
File: tasks.conditional_mass_function.F90
Used by:

Name: massHaloBinsPerDecade
Type: real 0..1
Default value: 10
Description: The number of bins per decade of halo mass to use when constructing the mass function covariance matrix for main branch galaxies.
Defined in: function:correlationFunctionHearin2013SDSSConstructorParameters
File: galacticus.output.analyses.correlation_function.Hearin2014_SDSS.F90
Used by:

Name: massHaloDeclineFactor
Type: real 1
Default value: 0.9
Description: The factor by which halo mass should decrease in each step back in time building a smoothly accreting merger tree, in units of $M_\odot$.
Defined in: function:smoothAccretionConstructorParameters
File: merger_trees.construct.smooth_accretion.F90
Used by:

Name: massHaloMaximum
Type: boolean 1
Default value: 1.0×10^{15}
Description: The minimum halo mass to use when computing mass function covariance matrices.
Defined in: function:massFunctionCovarianceConstructorParameters
File: tasks.mass_function_covariance.F90
Used by:

Name: massHaloMinimum
Type: float 1
Default value: 1.0×10^{10}
Description: The minimum halo mass to use when computing mass function covariance matrices.
Defined in: function:massFunctionCovarianceConstructorParameters
File: tasks.mass_function_covariance.F90
Used by:

Name: massHaloResolution
Type: real 1

Default value: 1.0×10^9
Description: The final mass of the merger tree base halo to consider when building a smoothly accreting merger tree, in units of $M_\odot$.
Defined in: function:smoothAccretionConstructorParameters
File: merger_trees.construct.smooth_accretion.F90
Used by:

Name: massInfiniteToMassSharpEdge
Type: real 0..1
Default value: 0.98 [More et al., 2011, estimate based on comments in text]
Description: The ratio of the friends-of-friends mass in the limit of infinite number of particles to the mass of the halo enclosed within a sharp-edged sphere bounding an isodensity surface equal to the critical density for percolation.
Defined in: function:fofBiasConstructorParameters
File: structure_formation.halo_mass_function.friends_of_friends_bias.F90
Used by:

Name: massIntervalFractional
Type: float 1
Default value: 0.1
Description: The fractional mass interval occupied by the trees. Where the intervals of trees of different mass would overlap this interval will be truncated.
Defined in: function:readXMLConstructorParameters
File: merger_trees.construct.build.masses.read.XML.F90
Used by:

Name: massLoading
Type: real 0..1
Default value: 2.0
Description: The mass-loading of "superwind" expulsive outflows in spheroids..
Defined in: function:superWindConstructorParameters
File: star_formation.feedback_expulsion.spheroids.superwind.F90
Used by:

Name: massLogarithmDelta
Type: real 1
Description: Logarithmic widths of mass bins for conditional mass function calculations.
Defined in: function:conditionalMassFunctionConstructorParameters
File: tasks.conditional_mass_function.F90
Used by:

Name: massLongLived
Type: real 1
Default value: 1.0
Description: The mass below which stars are assumed to be infinitely long-lived in the instantaneous approximation for stellar evolution.
Defined in: function:standardConstructorParameters
File: stellar_populations.standard.F90
Used by:

Name: massLower
Type: real 1
Default value: 0.1
Description: The lower mass limit for the Chabrier [2001] IMF.
Defined in: function:chabrier2001ConstructorParameters
File: stellar_populations.initial_mass_functions.Chabrier2001.F90
Used by:

Name: massMaxima
Type: float 0..1
Description: The maximum mass of each mass sample.
Defined in: function:correlationFunctionConstructorParameters
File: galacticus.output.analyses.correlation_function.F90
Used by:

Name: massMaximum
Type: float 1
Default value: 1.0×10^{13}
Description: The maximum mass in the mass function for covariance calculations.
Defined in: function:massFunctionCovarianceConstructorParameters
File: tasks.mass_function_covariance.F90
Used by:

Name: massMinima
Type: float 0..1
Description: The minimum mass of each mass sample.
Defined in: function:correlationFunctionConstructorParameters
File: galacticus.output.analyses.correlation_function.F90
Used by:

Name: massMinimum
Type: float 1
Default value: 1.0×10^{08}
Description: The minimum mass in the mass function for covariance calculations.
Defined in: function:massFunctionCovarianceConstructorParameters
File: tasks.mass_function_covariance.F90
Used by:

Name: massOptions
Type: string 1
Default value: var_str('default')
Description: Mass option for Sussing merger trees.
Defined in: function:sussingConstructorParameters
File: merger_trees.construct.read.importer.SussingMergerTrees.F90
Used by:

Name: massOvershootAttemptsMaximum
Type: integer 0..

Default value: 50
Description: The number of failed trees to allow before increasing the mass of the parent node.
Defined in: function:augmentConstructorParameters
File: merger_trees.operators.augment.F90
Used by:

Name: massOvershootScaleFactor
Type: real 0..
Default value: 0.05
Description: The amount by which to increase the mass overshoot factor after exhausting tree build attempts.
Defined in: function:augmentConstructorParameters
File: merger_trees.operators.augment.F90
Used by:

Name: massParticle
Type: real 1
Description: The mass of the simulation particle.
Defined in: function:mergerTreeFileBuilderConstructorParameters
File: tasks.merger_tree_file_builder.F90
Used by:

Name: massRangeMinimum
Type: real 1
Description: The minimum halo mass to include in the likelihood evaluation.
Defined in: function:haloMassFunctionConstructorParameters
File: models.likelihoods.halo_mass_function.F90
Used by:

Name: massRatioCount
Type: integer 1
Default value: 10
Description: The number of bins in mass ratio when constructing conditional halo mass functions.
Defined in: function:conditionalMFConstructorParameters
File: merger_trees.operators.conditional_mass_function.F90
Used by:

Name: massRatioMajorMerger
Type: real 1
Default value: 0.25
Description: The mass ratio above which mergers are considered to be "major".
Defined in: function:verySimpleConstructorParameters
File: satellites.merging.mass_movements.very_simple.F90
Used by:

Name: massRatioMaximum
Type: logical 1
Default value: 1.0×10^{1}
Description: The maximum mass ratio to bin when constructing conditional halo mass functions.

Defined in: function:conditionalMFConstructorParameters
File: merger_trees.operators.conditional_mass_function.F90
Used by:

Name: massRatioMinimum
Type: real 1
Default value: 1.0×10^{-4}
Description: The minimum mass ratio to bin when constructing conditional halo mass functions.
Defined in: function:conditionalMFConstructorParameters
File: merger_trees.operators.conditional_mass_function.F90
Used by:

Name: massResolution
Type: real 0..1
Default value: 5.0×10^{9}
Description: The mass resolution to use when building merger trees.
Defined in: function:fixedConstructorParameters
File: merger_trees.construct.build.mass_resolution.fixed.F90
Used by:

Name: massResolutionFractional
Type: real 0..1
Default value: 1.0×10^{-3}
Description: The fraction of the tree's root node mass to be used for the mass resolution when building merger trees.
Defined in: function:scaledConstructorParameters
File: merger_trees.construct.build.mass_resolution.scaled.F90
Used by:

Name: massResolutionMaximum
Type: real 0..1
Default value: huge(0.0d0)
Description: The maximum mass resolution to use when building merger trees.
Defined in: function:scaledConstructorParameters
File: merger_trees.construct.build.mass_resolution.scaled.F90
Used by:

Name: massResolutionMinimum
Type: real 0..1
Default value: 5.0×10^{9}
Description: The minimum mass resolution to use when building merger trees.
Defined in: function:scaledConstructorParameters
File: merger_trees.construct.build.mass_resolution.scaled.F90
Used by:

Name: massStellarRatio
Type: real 0..1
Default value: 0.3
Description: The stellar mass bulge-to-total ratio used to discriminate late-type vs. early-type galaxies.

Defined in: function:galaxySizesSDSSConstructorParameters
File: galacticus.output.analyses.galaxy_sizes_SDSS.F90
Used by:

Name: massThreshold
Type: real 0..1
Default value: 0.0
Description: Threshold mass below which merger tree branches should be pruned.
Defined in: function:pruneByMassConstructorParameters
File: merger_trees.operators.prune_by_mass.F90
Used by:

Name: massThresholdHarrassment
Type: real 1
Default value: 0.0
Description: The host halo mass threshold for harrassment to take effect.
Defined in: function:efstathiou1982TidalConstructorParameters
File: galactic_dynamics.bar_instability.Efstathiou1982Tidal.F90
Used by:

Name: massTransition
Type: real 1
Default value: 1.0
Description: The transition limit for the Chabrier [2001] IMF.
Defined in: function:chabrier2001ConstructorParameters
File: stellar_populations.initial_mass_functions.Chabrier2001.F90
Used by:

Name: massTree
Type: float 1
Default value: spread(1.0d12,1,fixedHalosCount)
Description: Specifies the masses of halos to use when building halos.
Defined in: function:fixedMassConstructorParameters
File: merger_trees.construct.build.masses.fixed_mass.F90
Used by:

Name: massTreeMaximum
Type: real 1
Default value: 1.0×10^{15}
Description: The maximum mass of merger tree base halos to consider when sampled masses from a distribution, in units of $M_\odot$.
Defined in: function:sampledDistributionConstructorParameters
File: merger_trees.construct.build.masses.sampled_distribution.F90
Used by:

Name: massTreeMinimum
Type: real 1
Default value: 1.0×10^{10}
Description: The minimum mass of merger tree base halos to consider when sampled masses from a

distribution, in units of $M_\odot$.
Defined in: function:sampledDistributionConstructorParameters
File: merger_trees.construct.build.masses.sampled_distribution.F90
Used by:

Name: massUpper
Type: real 1
Default value: 125.0
Description: The upper mass limit for the Chabrier [2001] IMF.
Defined in: function:chabrier2001ConstructorParameters
File: stellar_populations.initial_mass_functions.Chabrier2001.F90
Used by:

Name: masses
Type: float 0..1
Description: The masses corresponding to bin centers.
Defined in: function:massFunctionStellarConstructorParameters
File: galacticus.output.analyses.mass_function_stellar.F90
Used by:

Name: massesPerDecade
Type: integer 1
Default value: 10
Description: The number of points per decade of mass at which to tabulate excursion set solutions.
Defined in: function:excursionSetsConstructorParameters
File: tasks.excursion_sets.F90
Used by:

Name: maximinTrialCount
Type: integer 1
Default value: 1000
Description: The number of trial Latin Hypercubes to construct when seeking the maximum minimum separation sample.
Defined in: function:latinHypercubeConstructorParameters
File: posterior_sampling.state.initialize.latin_hypercube.F90
Used by:

Name: mean
Type: real 1
Description: The mean of the normal distribution.
Defined in: function:normalConstructorParameters
File: statistics.distributions.normal.F90
Used by:

Name: meanComment
Type: string 0..1
Description: A descriptive comment for the mean.
Defined in: function:meanFunction1DConstructorParameters
File: galacticus.output.analyses.mean_function_1d.F90

Used by:

Name: `meanCovarianceTarget`
Type: real 0..1
Description: The target function covariance for likelihood calculations.
Defined in: `function:meanFunction1DConstructorParameters`
File: `galacticus.output.analyses.mean_function_1d.F90`
Used by:

Name: `meanLabel`
Type: string 0..1
Description: A label for the mean.
Defined in: `function:meanFunction1DConstructorParameters`
File: `galacticus.output.analyses.mean_function_1d.F90`
Used by:

Name: `meanUnits`
Type: string 0..1
Description: A human-readable description of the units for the mean.
Defined in: `function:meanFunction1DConstructorParameters`
File: `galacticus.output.analyses.mean_function_1d.F90`
Used by:

Name: `meanUnitsInSI`
Type: string 0..1
Description: A units for the mean in the SI system.
Defined in: `function:meanFunction1DConstructorParameters`
File: `galacticus.output.analyses.mean_function_1d.F90`
Used by:

Name: `meanValueTarget`
Type: real 0..1
Description: The target function for likelihood calculations.
Defined in: `function:meanFunction1DConstructorParameters`
File: `galacticus.output.analyses.mean_function_1d.F90`
Used by:

Name: `means`
Type: real 1
Description: The mean of the multivariate normal distribution.
Defined in: `function:multivariateNormalStochasticConstructorParameters`
File: `models.likelihoods.multivariate_normal.stochastic.F90`
Used by:

Name: `median`
Type: real 1
Description: The median of the Cauchy distribution function.
Defined in: `function:cauchyConstructorParameters`
File: `statistics.distributions.Cauchy.F90`

Used by:

Name: `mergeProbability`
Type: real 0..1
Default value: 0.1
Description: The largest probability of branching allowed in a timestep in merger trees built by the Cole et al. [2000] method.
Defined in: `function:cole2000ConstructorParameters`
File: `merger_trees.construct.builder.Cole2000.F90`
Used by:

Name: `mergerTreeEvolutionDump`
Type: boolean 1
Default value: .false.
Description: Specifies whether or not to output the evolution of merger trees.
Defined in: `subroutine:Merger_Tree_Dump_Evolution`
File: `merger_trees.dump_evolution.F90`
Used by:

Name: `mergerTreeEvolutionDumpFileName`
Type: text 1
Default value: var_str('mergerTreeEvolution.xml')
Description: Specifies the file to which merger tree evolution should be dumped.
Defined in: `subroutine:Merger_Tree_Dump_Evolution`
File: `merger_trees.dump_evolution.F90`
Used by:

Name: `mergerTreeStructureDump`
Type: boolean 1
Default value: .false.
Description: Specifies whether or not to output the structure of merger trees prior to evolution.
Defined in: `subroutine:Merger_Tree_Structure_Dump`
File: `merger_trees.dump_structure.F90`
Used by:

Name: `mergerTreeStructureDumpDirectory`
Type: text 1
Default value: var_str('.')
Description: Specifies the directory to which merger tree structure should be dumped.
Defined in: `subroutine:Merger_Tree_Structure_Dump`
File: `merger_trees.dump_structure.F90`
Used by:

Name: `mergerTreeStructureDumpMassMaximum`
Type: real 1
Default value: huge(1.0d0)
Description: Specifies the minimum root mass for which merger tree structure should be dumped.
Defined in: `subroutine:Merger_Tree_Structure_Dump`
File: `merger_trees.dump_structure.F90`

Used by:

Name: `mergerTreeStructureDumpMassMinimum`
Type: real 1
Default value: 0.0
Description: Specifies the minimum root mass for which merger tree structure should be dumped.
Defined in: subroutine:Merger_Tree_Structure_Dump
File: merger_trees.dump_structure.F90
Used by:

Name: `mergerTreeStructureOutput`
Type: boolean 1
Default value: .false.
Description: Specifies whether or not to output the structure of merger trees prior to evolution.
Defined in: subroutine:Merger_Tree_Structure_Output
File: merger_trees.output_structure.F90
Used by:

Name: `mergerTreeStructureOutputDarkMatterProfileScale`
Type: boolean 1
Default value: .false.
Description: Determines whether or not dark matter halo scale radius is included in outputs of merger trees.
Defined in: subroutine:Node_Component_Dark_Matter_Profile_Scale_Initialize
File: objects.nodes.components.dark_matter_profile.scale.F90
Used by:

Name: `mergerTreeStructureOutputDarkMatterProfileShape`
Type: boolean 1
Default value: .false.
Description: Determines whether or not dark matter halo shape parameter is included in outputs of merger trees.
Defined in: subroutine:Node_Component_Dark_Matter_Profile_Scale_Shape_Initialize
File: objects.nodes.components.dark_matter_profile.scale_shape.F90
Used by:

Name: `mergerTreeStructureOutputVirialQuantities`
Type: boolean 1
Default value: .false.
Description: Specifies whether or not to output virial quantities (radius and velocity) when outputting the structure of merger trees prior to evolution.
Defined in: subroutine:Merger_Tree_Structure_Output
File: merger_trees.output_structure.F90
Used by:

Name: `metaCollectMemoryUsageData`
Type: boolean 1
Default value: .false.
Description: Specifies whether or not collect and output data on the memory used while processing

trees.
Defined in: subroutine:Meta_Tree_Timing_Initialize
File: galacticus.meta.tree_timing.F90
Used by:

Name: metaCollectTimingData
Type: boolean 1
Default value: .false.
Description: Specifies whether or not collect and output data on the time spent processing trees.
Defined in: subroutine:Meta_Tree_Timing_Initialize
File: galacticus.meta.tree_timing.F90
Used by:

Name: metaProfileTimeStepMaximum
Type: real 1
Default value: 1.0×10^{1}
Description: The largest timestep to use in profiling ODE solver steps.
Defined in: subroutine:Galacticus_Meta_Evolver_Profile
File: galacticus.meta.evolver_profiler.F90
Used by:

Name: metaProfileTimeStepMinimum
Type: real 1
Default value: 1.0×10^{-6}
Description: The smallest timestep to use in profiling ODE solver steps.
Defined in: subroutine:Galacticus_Meta_Evolver_Profile
File: galacticus.meta.evolver_profiler.F90
Used by:

Name: metaProfileTimeStepPointsPerDecade
Type: integer 1
Default value: 3
Description: The number of bins per decade of timestep to use when profiling ODE solver steps.
Defined in: subroutine:Galacticus_Meta_Evolver_Profile
File: galacticus.meta.evolver_profiler.F90
Used by:

Name: metalYield
Type: real 1
Default value: 0.0
Description: The metal yield to use in the instantaneous stellar evolution approximation. (If not specified it will be computed internally.)
Defined in: function:standardConstructorParameters
File: stellar_populations.standard.F90
Used by:

Name: metallicityBoundaries
Type: real 1..
Description: The metallicities corresponding to boundaries between metallicity bins to use when tabu-

lating star formation histories.
Defined in: function:metallicitySplitConstructorParameters
File: star_formation.histories.metallicity_split.F90
Used by:

Name: metallicityMaximum
Type: real 1
Default value: 1.0×10^{1}
Description: The upper limit to the metallicity in the highest metallicity bin when tabulating star formation histories [Solar units].
Defined in: function:metallicitySplitConstructorParameters
File: star_formation.histories.metallicity_split.F90
Used by:

Name: metallicityMinimum
Type: real 1
Default value: 1.0×10^{-4}
Description: The upper limit to the metallicity in the lowest metallicity bin when tabulating star formation histories [Solar units].
Defined in: function:metallicitySplitConstructorParameters
File: star_formation.histories.metallicity_split.F90
Used by:

Name: metallicitySystematicErrorPolynomialCoefficient
Type: float 0..1
Default value: [0.0d0]
Description: The coefficients of the metallicity systematic error polynomial.
Defined in: function:massMetallicityBlanc2017ConstructorParameters
File: galacticus.output.analyses.mass_metallicity_relation.Blanc2017.F90
Used by:

Name: missingHostsAreFatal
Type: boolean 1
Default value: .true.
Description: Specifies whether nodes with missing host nodes should be considered to be fatal—see §5.2.1.
Defined in: function:readConstructorParameters
File: merger_trees.construct.read.F90
Used by:

Name: model
Type: string 1
Default value: var_str('MilkyWayShellTau3.0')
Description: The name of the model from Witt and Gordon [2000] to use in dust attenuation calculations.
Defined in: function:wittGordon2003ConstructorParameters
File: stellar_populations.spectra.dust_attenuation.WittGordon2000.F90
Used by:

Name: `modelParameterName1`
Type: string 0..
Description: Names of parameters to be initialized by initializer number 1.
Defined in: function:switchedConstructorParameters
File: posterior_sampling.state.initialize.switched.F90
Used by:

Name: `modelParameterName2`
Type: string 0..
Description: Names of parameters to be initialized by initializer number 2.
Defined in: function:switchedConstructorParameters
File: posterior_sampling.state.initialize.switched.F90
Used by:

Name: `modelSurfaceBrightness`
Type: boolean 1
Description: If true, model the effects of surface brightness incompleteness on the mass function.
Defined in: function:massFunctionConstructorParameters
File: models.likelihoods.mass_function.F90
Used by:

Name: `modifier1`
Type: string 1
Default value: 0.0
Description: Coefficient of the polynomial modifier applied to the halo mass function when sampling halo masses for tree construction.
Defined in: function:haloMassFunctionConstructorParameters
File: merger_trees.construct.build.masses.distribution.halo_mass_function.F90
Used by:

Name: `modifier2`
Type: string 1
Default value: 0.0
Description: Coefficient of the polynomial modifier applied to the halo mass function when sampling halo masses for tree construction.
Defined in: function:haloMassFunctionConstructorParameters
File: merger_trees.construct.build.masses.distribution.halo_mass_function.F90
Used by:

Name: `molecularFractionFast`
Type: boolean 1
Default value: .false.
Description: Selects whether the fast (but less accurate) fitting formula for molecular hydrogen should be used in the "Krumholz-McKee-Tumlinson" star formation timescale calculation.
Defined in: function:krumholz2009ConstructorParameters
File: star_formation.rate_surface_density.disks.Krumholz2009.F90
Used by:

Name: `monotonicInterpolation`

Type: real 1
Default value: .false.
Description: If true use a monotonic cubic spline interpolator to interpolate in the $\sigma(M)$ table. Otherwise use a standard cubic spline interpoltor. Use of the monotionic interpolator can be helpful is $\sigma(M)$ must be strictly monotonic but because a very weak function of M at low masses.
Defined in: `function:filteredPowerConstructorParameters`
File: `structure_formation.cosmological_mass_variance.filtered_power_spectrum.F90`
Used by:

Name: `mu`
Type: real 1
Description: The parameter γ of the Voight function.
Defined in: `function:voightConstructorParameters`
File: `statistics.distributions.Voight.F90`
Used by:

Name: `muRatioHigh`
Type: real 1
Default value: [+1.100d0,+1.100d0,+1.084d0]
Description: Values of the μ parameter of the Jiang et al. [2014] orbital velocity distribution for the three host halo mass ranges, and the 0.05–0.5 mass ratio.
Defined in: `function:jiang2014ConstructorParameters`
File: `satellites.merging.virial_orbits.Jiang2014.F90`
Used by:

Name: `muRatioIntermediate`
Type: real 1
Default value: [+1.181d0,+1.201d0,+1.236d0]
Description: Values of the μ parameter of the Jiang et al. [2014] orbital velocity distribution for the three host halo mass ranges, and the 0.005–0.05 mass ratio.
Defined in: `function:jiang2014ConstructorParameters`
File: `satellites.merging.virial_orbits.Jiang2014.F90`
Used by:

Name: `muRatioLow`
Type: real 1
Default value: [+1.220d0,+1.231d0,+1.254d0]
Description: Values of the μ parameter of the Jiang et al. [2014] orbital velocity distribution for the three host halo mass ranges, and the 0.0001–0.005 mass ratio.
Defined in: `function:jiang2014ConstructorParameters`
File: `satellites.merging.virial_orbits.Jiang2014.F90`
Used by:

Name: `multiplier`
Type: float 0..1
Description: Multiplying factor.
Defined in: `function:multiplyConstructorParameters`
File: `galacticus.output.analyses.property_operator.multiply.F90`
Used by:

Name: nBodyFileName
Type: string 1
Description: The name of the file containing the N-body data.
Defined in: function:nbodyAnalyzeConstructorParameters
File: tasks.nBody_analyze.F90
Used by:

Name: name
Type: string 1
Description: A human-readable name for the units of velocity.
Defined in: function:mergerTreeFileBuilderConstructorParameters
File: tasks.merger_tree_file_builder.F90
Used by:

Name: names
Type: string 1
Default value: [var_str('default')]
Description: The names assigned to stellar spectra postprocessors.
Defined in: function:lookupConstructorParameters
File: stellar_populations.spectra.postprocess.builder.lookup.F90
Used by:

Name: nbodyFileNamePrevious
Type: string 1
Description: The name of the file containing the N-body data.
Defined in: function:nbodyAnalyzeConstructorParameters
File: tasks.nBody_analyze.F90
Used by:

Name: negativeBinomialScatterFractional
Type: float 0..1
Default value: 0.18 [Boylan-Kolchin et al., 2010]
Description: The fractional scatter (relative to the Poisson scatter) in the negative binomial distribution used in likelihood calculations.
Defined in: function:localGroupMassFunctionConstructorParameters
File: galacticus.output.analyses.Local_Group_mass_functions.F90
Used by:

Name: neighborCount
Type: string 1
Description: The number of nearest neighbors to use when estimating the posterior likelihood.
Defined in: function:posteriorAsPriorConstructorParameters
File: models.likelihoods.posterior_as_prior.F90
Used by:

Name: neutrinoMassSummed
Type: real 0..1
Default value: 0.0

Description: The summed mass (in electron volts) of all neutrino species.
Defined in: function:eisensteinHu1999ConstructorParameters
File: structure_formation.transfer_function.Eisenstein_Hu.F90
Used by:

Name: neutrinoNumberEffective
Type: real 0..1
Default value: 3.046 [Mangano et al., 2005]
Description: The effective number of neutrino species.
Defined in: function:eisensteinHu1999ConstructorParameters
File: structure_formation.transfer_function.Eisenstein_Hu.F90
Used by:

Name: nodeComponentBasicExtendedSphericalCollapseEnergyFixedAt
Type: string 1
Default value: var_str('turnaround')
Description: Selects the epoch at which the energy of a spherical top hat perturbation in a dark energy cosmology should be "fixed" for the purposes of computing virial density contrasts. (See the discussion in Percival 2005; §8.).
Defined in: subroutine:Node_Component_Basic_Extended_Bindings
File: objects.nodes.components.basic.extended.F90
Used by:

Name: nodeComponentBasicExtendedSphericalCollapseType
Type: string 1
Default value: var_str('matterLambda')
Description: The type of spherical collapse model to assume in the extended basic node component class.
Defined in: subroutine:Node_Component_Basic_Extended_Bindings
File: objects.nodes.components.basic.extended.F90
Used by:

Name: nodeFormationMassFraction
Type: double 1
Default value: 0.5
Description: The mass fraction in the main branch progenitor used to define the formation time of each halo.
Defined in: subroutine:Node_Component_Merging_Statistics_Standard_Initialize
File: objects.nodes.components.merging_statistics.standard.F90
Used by:

Name: nodeMajorMergerFraction
Type: double 1
Default value: 0.25
Description: The mass ratio (M_2/M_1 where $M_2 < M_1$) of merging halos above which the merger should be considered to be "major".
Defined in: subroutine:Node_Component_Merging_Statistics_Standard_Initialize
File: objects.nodes.components.merging_statistics.standard.F90
Used by:

Name: nodePromotionIndexShift
Type: boolean 1
Default value: .false.
Description: Specifies whether or not the index of a node should be shifted to its parent node prior to promotion.
Defined in: subroutine:Node_Promotion_Index_Shift
File: events.node_promotion.index_shift.F90
Used by:

Name: nodeRecentMajorMergerFromInfall
Type: double 1
Default value: .false.
Description: Specifies whether "recent" for satellite galaxies is measured from the current time, or from the time at which they were last isolated.
Defined in: subroutine:Node_Component_Merging_Statistics_Recent_Initialize
File: objects.nodes.components.merging_statistics.recent.F90
Used by:

Name: nodeRecentMajorMergerInterval
Type: double 1
Default value: 2.0
Description: The time interval used to define "recent" mergers in the `recent` merging statistics component. This parameter is in units of Gyr if `[nodeRecentMajorMergerIntervalType]`=`absolute`, or in units of the halo dynamical time if `[nodeRecentMajorMergerIntervalType]`=`dynmical`.
Defined in: subroutine:Node_Component_Merging_Statistics_Recent_Initialize
File: objects.nodes.components.merging_statistics.recent.F90
Used by:

Name: nodeRecentMajorMergerIntervalType
Type: double 1
Default value: var_str('dynamical')
Description: Specifies the units for the `[nodeRecentMajorMergerInterval]` parameter. If set to `absolute` then `[nodeRecentMajorMergerInterval]` is given in Gyr, while if set to `dynamical` `[nodeRecentMajorMergerInte` is given in units of the halo dynamical time.
Defined in: subroutine:Node_Component_Merging_Statistics_Recent_Initialize
File: objects.nodes.components.merging_statistics.recent.F90
Used by:

Name: nonAnalyticSolver
Type: string 1
Default value: var_str('fallThrough')
Description: Selects how solutions are computed when no analytic solution is available. If set to "`fallThrough`" then the solution ignoring heating is used, while if set to "`numerical`" then numerical solvers are used to find solutions.
Defined in: function:truncatedExponentialConstructorParameters
File: dark_matter_profiles_DMO.truncated.exponential.F90
Used by:

Name: `nonCosmological`
Type: boolean 0..1
Default value: .false.
Description: If true, a non-cosmological snapshot file will be created.
Defined in: function:particulateConstructorParameters
File: merger_trees.operators.particulate.F90
Used by:

Name: `normalization`
Type: string 1
Default value: var_str('natural')
Description: The parameter a in the relation $k_\mathrm{s} = a/r_\mathrm{s}$, where k_s is the cut-off wavenumber for the sharp k-space window function and r_s is the radius of a sphere (in real-space) enclosing the requested smoothing mass. Alternatively, a value of `natural` will be supplied in which case the normalization is chosen such that, in real-space, $W(r=0)=1$. This results in a contained mass of $M = 6\pi^2\bar{\rho}k_\mathrm{s}^{-3}$.
Defined in: function:topHatSharpKHybridConstructorParameters
File: structure_formation.power_spectrum.variance.window_function.top_hat_kspace_sharp_-hybrid.F90
Used by:

Name: `nu`
Type: real 1
Default value: 1.1 [Barkana et al., 2001, for the transfer function at $z = z_\mathrm{eq}$]
Description: The parameter ϵ appearing in the warm dark matter transfer function [Barkana et al., 2001].
Defined in: function:bode2001ConstructorParameters
File: structure_formation.transfer_function.Bode2001.F90
Used by:

Name: `odeAlgorithm`
Type: real 1
Default value: var_str('rungeKuttaCashKarp')
Description: The algorithm to use in the ODE solver.
Defined in: function:standardConstructorParameters
File: merger_trees.node_evolver.standard.F90
Used by:

Name: `odeAlgorithmNonJacobian`
Type: real 1
Default value: var_str('rungeKuttaCashKarp')
Description: The algorithm to use in the ODE solver.
Defined in: function:standardConstructorParameters
File: merger_trees.node_evolver.standard.F90
Used by:

Name: `odeJacobianStepSizeRelative`
Type: real 1
Default value: 0.01
Description: The relative step size to use when perturbing properties for purposes of computing a finite

difference approximation to the ODE system Jacobian.
Defined in: function:standardConstructorParameters
File: merger_trees.node_evolver.standard.F90
Used by:

Name: odeLatentIntegratorIntervalsMaximum
Type: integer 1
Default value: 1000
Description: The maxium number of intervals allowed in the integrator for latent variables.
Defined in: function:standardConstructorParameters
File: merger_trees.node_evolver.standard.F90
Used by:

Name: odeLatentIntegratorOrder
Type: integer 1
Default value: 15
Description: The order of the integrator for latent variables.
Defined in: function:standardConstructorParameters
File: merger_trees.node_evolver.standard.F90
Used by:

Name: odeLatentIntegratorType
Type: string 1
Default value: var_str('trapezoidal')
Description: The type of integrator to use for latent variables.
Defined in: function:standardConstructorParameters
File: merger_trees.node_evolver.standard.F90
Used by:

Name: odeToleranceAbsolute
Type: real 1
Default value: 0.01
Description: The absolute tolerance used in solving differential equations for node evolution.
Defined in: function:standardConstructorParameters
File: merger_trees.node_evolver.standard.F90
Used by:

Name: odeToleranceRelative
Type: real 1
Default value: 1.0×10^{-2}
Description: The relative tolerance used in solving differential equations for node evolution.
Defined in: function:standardConstructorParameters
File: merger_trees.node_evolver.standard.F90
Used by:

Name: offset
Type: integer 1
Description: The offset of the stride to take over forests.
Defined in: function:strideConstructorParameters

File: tasks.evolve_forests.work_share.stride.F90
Used by:

Name: omega
Type: real 1
Default value: 0.77 (Gustafsson et al. 2006; from their Fig. 9, strong feedback case)
Description: The parameter ω appearing in the Gnedin et al. [2004] adiabatic contraction algorithm.
Defined in: function:adiabaticGnedin2004ConstructorParameters
File: dark_matter_profiles.adiabatic_Gnedin2004.F90
Used by:

Name: opacityExponent
Type: real 1
Default value: 0.7
Description: The exponent of wavelength appearing in the opacity.
Defined in: function:charlotFall2000ConstructorParameters
File: stellar_populations.spectra.dust_attenuation.CharlotFall2000.F90
Used by:

Name: opticalDepthBirthClouds
Type: real 1
Default value: 1.0
Description: The effective optical depth of birth clouds.
Defined in: function:charlotFall2000ConstructorParameters
File: stellar_populations.spectra.dust_attenuation.CharlotFall2000.F90
Used by:

Name: opticalDepthISM
Type: real 1
Default value: 0.5
Description: The effective optical depth of the ISM.
Defined in: function:charlotFall2000ConstructorParameters
File: stellar_populations.spectra.dust_attenuation.CharlotFall2000.F90
Used by:

Name: opticalDepthReionization
Type: real 1
Description: The optical depth to electron scattering below which baryonic accretion is suppressed.
Defined in: function:simpleConstructorParameters
File: accretion.halo.simple.F90
Used by:

Name: origin
Type: float 1
Default value: [randomSequence%uniformSample(),randomSequence%uniformSample(),randomSequence%uniformSample(
Uniformly random distribution within the box.
Description: The vector (in units of the box length) giving the origin of the coordinate system to use in mock catalog construction.
Defined in: function:catalogProjectedCorrelationFunctionConstructorParameters

File: tasks.catalog_projected_correlation_function.F90
Used by:

Name: outerRadius
Type: real 1
Description: The outer radius of a β-model mass distribution.
Defined in: function:betaProfileConstructorParameters
File: mass_distributions.spherical.beta_profile.F90
Used by:

Name: outlierCountMaximum
Type: integer 1
Default value: 0
Description: The maximum number of outlier states allowed.
Defined in: function:gelmanRubinConstructorParameters
File: posterior_sampling.convergence.Gelman-Rubin.F90
Used by:

Name: outlierLogLikelihoodOffset
Type: real 1
Default value: 0.0
Description: The log-likelihood offset at which to declare a state an outlier.
Defined in: function:gelmanRubinConstructorParameters
File: posterior_sampling.convergence.Gelman-Rubin.F90
Used by:

Name: outlierSignificance
Type: real 1
Default value: 0.05
Description: The significance at which to declare a state an outlier.
Defined in: function:gelmanRubinConstructorParameters
File: posterior_sampling.convergence.Gelman-Rubin.F90
Used by:

Name: outputFileName
Type: string 1
Description: The name of the file to which to write merger tree data.
Defined in: function:mergerTreeFileBuilderConstructorParameters
File: tasks.merger_tree_file_builder.F90
Used by:

Name: outputFormat
Type: string 1
Description: The format to use for merger tree output.
Defined in: function:mergerTreeFileBuilderConstructorParameters
File: tasks.merger_tree_file_builder.F90
Used by:

Name: outputGroup

Type: integer 1
Default value: var_str('.')
Description: The HDF5 output group within which to write power spectrum data.
Defined in: function:powerSpectraConstructorParameters
File: tasks.power_spectrum.F90
Used by:

Name: outputGroupName
Type: string 1
Description: The name of the file to which the computed conditional mass function should be output.
Defined in: function:conditionalMassFunctionConstructorParameters
File: tasks.conditional_mass_function.F90
Used by:

Name: outputHaloModelData
Type: boolean 1
Default value: .false.
Description: Specifies whether or not halo model data (bias, power spectra, etc.) should be included in the output.
Defined in: subroutine:Galacticus_Output_Halo_Model_Initialize
File: galacticus.output.merger_tree.halo_model.F90
Used by:

Name: outputReferences
Type: boolean 1
Default value: .false.
Description: Specifies whether or not references to individual merger tree datasets should be output.
Defined in: function:standardConstructorParameters
File: merger_trees.outputter.standard.F90
Used by:

Name: outputSatelliteMergers
Type: boolean 1
Default value: .false.
Description: Specifies whether satellite merger information should be output.
Defined in: subroutine:Satellite_Merging_Output
File: satellites.merging.output.F90
Used by:

Name: outputSatelliteMergersMainBranchOnly
Type: boolean 1
Default value: .true.
Description: Specifies whether satellite merger information should be output only for the main branch host halo.
Defined in: subroutine:Satellite_Merging_Output
File: satellites.merging.output.F90
Used by:

Name: outputTimeSnapTolerance

Type: real 1
Default value: 0.0
Description: The relative tolerance required to "snap" a node time to the closest output time.
Defined in: function:readConstructorParameters
File: merger_trees.construct.read.F90
Used by:

Name: outputWeight
Type: float 0..1
Description: The weight to assign to each bin at each output.
Defined in: function:volumeFunction1DConstructorParameters
File: galacticus.output.analyses.volume_function_1d.F90
Used by:

Name: outputsGroupName
Type: string 1
Default value: var_str('Outputs')
Description: The name of the HDF5 group to which outputs will be written.
Defined in: function:standardConstructorParameters
File: merger_trees.outputter.standard.F90
Used by:

Name: p
Type: real 0..1
Default value: 0.3
Description: The parameter p in the Sheth et al. [2001] halo mass function fit.
Defined in: function:shethTormenConstructorParameters
File: structure_formation.halo_mass_function.Sheth-Tormen.F90
Used by:

Name: parentMassCount
Type: integer 1
Default value: 10
Description: The number of bins in parent mass when constructing conditional halo mass functions.
Defined in: function:conditionalMFConstructorParameters
File: merger_trees.operators.conditional_mass_function.F90
Used by:

Name: parentMassMaximum
Type: real 1
Default value: 1.0×10^{15}
Description: The maximum parent halo mass to bin when constructing conditional halo mass functions.
Defined in: function:conditionalMFConstructorParameters
File: merger_trees.operators.conditional_mass_function.F90
Used by:

Name: parentMassMinimum
Type: real 1
Default value: 1.0×10^{10}

Description: The minimum parent halo mass to bin when constructing conditional halo mass functions.
Defined in: function:conditionalMFConstructorParameters
File: merger_trees.operators.conditional_mass_function.F90
Used by:

Name: parentRedshifts
Type: real 1..
Default value: [0.0d0]
Description: The set of parent halo redshifts to use when constructing conditional halo mass functions.
Defined in: function:conditionalMFConstructorParameters
File: merger_trees.operators.conditional_mass_function.F90
Used by:

Name: particleCountMinimum
Type: real 1
Description: The minimum particle count used in N-body halos.
Defined in: function:spinDistributionConstructorParameters
File: models.likelihoods.spin_distribution.F90
Used by:

Name: particleType
Type: integer 0..1
Description: The particle type to read from the Gadget HDF5 file.
Defined in: function:gadgetHDF5ConstructorParameters
File: nBody.import.GadgetHDF5.F90
Used by:

Name: particlesFileName
Type: string 1
Description: The name of the file from which to read particle data.
Defined in: function:mergerTreeFileBuilderConstructorParameters
File: tasks.merger_tree_file_builder.F90
Used by:

Name: path
Type: text 1
Default value: var_str('.')
Description: Specifies the directory to which merger tree structure should be dumped.
Defined in: function:dumpToGraphVizConstructorParameters
File: merger_trees.operators.dump_to_GraphViz.F90
Used by:

Name: performChecks
Type: logical 0..
Default value: .false.
Description: If true, perform checks of the augmentation process.
Defined in: function:augmentConstructorParameters
File: merger_trees.operators.augment.F90
Used by:

Name: `periodic`
Type: real 0..1
Default value: .false.
Description: If true, periodic boundary conditions will be used.
Defined in: `function:environmentalOverdensityConstructorParameters`
File: `nBody.operator.environmental_overdensity.F90`
Used by:

Name: `phi0`
Type: real 1
Default value: 6.58
Description: The parameter ϕ_0 appearing in the halo concentration algorithm of Diemer and Kravtsov [2014].
Defined in: `function:diemerKravtsov2014ConstructorParameters`
File: `dark_matter_profiles.structure.concentration.Diemer-Kravtsov2014.F90`
Used by:

Name: `phi1`
Type: real 1
Default value: 1.37
Description: The parameter ϕ_1 appearing in the halo concentration algorithm of Diemer and Kravtsov [2014].
Defined in: `function:diemerKravtsov2014ConstructorParameters`
File: `dark_matter_profiles.structure.concentration.Diemer-Kravtsov2014.F90`
Used by:

Name: `pointsPerDecade`
Type: integer 1
Default value: 10
Description: The number of points per decade of wavenumber at which to tabulate power spectra.
Defined in: `function:powerSpectraConstructorParameters`
File: `tasks.power_spectrum.F90`
Used by:

Name: `polynomialOrder`
Type: integer 1
Default value: 2
Description: The order of the polynomial to fit to the likelihood surface.
Defined in: `function:gaussianRegressionConstructorParameters`
File: `models.likelihoods.gaussian_regression.F90`
Used by:

Name: `positionOffset`
Type: boolean 0..1
Default value: .false.
Description: If true, offset particle representations to the positions/velocities of nodes.
Defined in: `function:particulateConstructorParameters`
File: `merger_trees.operators.particulate.F90`

Used by:

Name: `positionsArePeriodic`
Type: boolean 1
Description: Specifies whether or not positions are periodic.
Defined in: `function:mergerTreeFileBuilderConstructorParameters`
File: `tasks.merger_tree_file_builder.F90`
Used by:

Name: `positionsPresetSatelliteToHost`
Type: bool 1
Default value: .false.
Description: If true, the position of satellite halos will be adjusted to match that of their host halo.
Defined in: `subroutine:Node_Component_Position_Preset_Initialize`
File: `objects.nodes.components.position.preset.F90`
Used by:

Name: `postprocessChain`
Type: string 0..1
Description: The postprocessing chain to use.
Defined in: `function:luminosityStellarConstructorParameters`
File: `nodes.property_extractor.luminosity_stellar.F90`
Used by:

Name: `preReionizationTemperature`
Type: real 1
Default value: 10.0
Description: The pre-reionization temperature (in units of Kelvin) in the simple IGM state model.
Defined in: `function:simpleIGMConstructorParameters`
File: `intergalactic_medium.state.simple.F90`
Used by:

Name: `precisionHypergeometric`
Type: real 1
Default value: 1.0×10^{-6}
Description: The fractional precision required in evaluates of hypergeometric functions in the modified Press-Schechter tree branching calculations.
Defined in: `function:parkinsonColeHellyConstructorParameters`
File: `merger_trees.branching_probability.Parkinson_Cole_Helly.F90`
Used by:

Name: `presentDayTemperature`
Type: real 1
Default value: 1.0×10^{3}
Description: The present day temperature (in units of Kelvin) in the instantReionization IGM state model.
Defined in: `function:instantReionizationIGMConstructorParameters`
File: `intergalactic_medium.state.instant_reionization.F90`
Used by:

Name: `preservePrimaryProgenitor`
Type: boolean 0..1
Default value: .true.
Description: If true, primary progenitor status is preserved even if the primary progenitor is pruned from the tree.
Defined in: function:pruneByMassConstructorParameters
File: merger_trees.operators.prune_by_mass.F90
Used by:

Name: `presetMergerNodes`
Type: boolean 1
Default value: .true.
Description: Specifies whether the target nodes for mergers should be preset (i.e. determined from descendent nodes). If they are not, merging will be with each satellite's host node.
Defined in: function:readConstructorParameters
File: merger_trees.construct.read.F90
Used by:

Name: `presetMergerTimes`
Type: boolean 1
Default value: .true.
Description: Specifies whether merging times for subhalos should be preset when reading merger trees from a file.
Defined in: function:readConstructorParameters
File: merger_trees.construct.read.F90
Used by:

Name: `presetNamedIntegers`
Type: string 1..
Description: Names of integer datasets to be additionally read and stored in the nodes of the merger tree when using the `[mergerTreeConstructMethod]=read` tree construction method.
Defined in: function:readConstructorParameters
File: merger_trees.construct.read.F90
Used by:

Name: `presetNamedReals`
Type: string 1..
Description: Names of real datasets to be additionally read and stored in the nodes of the merger tree when using the `[mergerTreeConstructMethod]=read` tree construction method.
Defined in: function:readConstructorParameters
File: merger_trees.construct.read.F90
Used by:

Name: `presetOrbits`
Type: boolean 1
Default value: .true.
Description: Specifies whether node orbits should be preset when reading merger trees from a file.
Defined in: function:readConstructorParameters

File: merger_trees.construct.read.F90
Used by:

Name: presetOrbitsAssertAllSet
Type: boolean 1
Default value: .true.
Description: Asserts that all virial orbits must be preset. If any can not be set, Galacticus will stop.
Defined in: function:readConstructorParameters
File: merger_trees.construct.read.F90
Used by:

Name: presetOrbitsBoundOnly
Type: boolean 1
Default value: .true.
Description: Specifies whether only bound node orbits should be set.
Defined in: function:readConstructorParameters
File: merger_trees.construct.read.F90
Used by:

Name: presetOrbitsSetAll
Type: boolean 1
Default value: .true.
Description: Forces all orbits to be set. If the computed orbit does not cross the virial radius, then select one at random instead.
Defined in: function:readConstructorParameters
File: merger_trees.construct.read.F90
Used by:

Name: presetPositions
Type: boolean 1
Default value: .true.
Description: Specifies whether node positions should be preset when reading merger trees from a file.
Defined in: function:readConstructorParameters
File: merger_trees.construct.read.F90
Used by:

Name: presetScaleRadii
Type: boolean 1
Default value: .true.
Description: Specifies whether node scale radii should be preset when reading merger trees from a file.
Defined in: function:readConstructorParameters
File: merger_trees.construct.read.F90
Used by:

Name: presetScaleRadiiConcentrationMaximum
Type: boolean 1
Default value: 60.0
Description: The largest concentration ($c = r_{\rm vir}/r_{\rm s}$) allowed when setting scale radii, $r_{\rm s}$.
Defined in: function:readConstructorParameters

File: merger_trees.construct.read.F90
Used by:

Name: presetScaleRadiiConcentrationMinimum
Type: boolean 1
Default value: 3.0
Description: The lowest concentration ($c = r_{\rm vir}/r_{\rm s}$) allowed when setting scale radii, $r_{\rm s}$.
Defined in: function:readConstructorParameters
File: merger_trees.construct.read.F90
Used by:

Name: presetScaleRadiiMinimumMass
Type: boolean 1
Default value: 0.0
Description: The minimum halo mass for which scale radii should be preset (if [presetScaleRadii]=true).
Defined in: function:readConstructorParameters
File: merger_trees.construct.read.F90
Used by:

Name: presetSpins
Type: boolean 1
Default value: .true.
Description: Specifies whether node spins should be preset when reading merger trees from a file.
Defined in: function:readConstructorParameters
File: merger_trees.construct.read.F90
Used by:

Name: presetSpins3D
Type: boolean 1
Default value: .false.
Description: Specifies whether node 3-D spin vectors should be preset when reading merger trees from a file.
Defined in: function:readConstructorParameters
File: merger_trees.construct.read.F90
Used by:

Name: presetSubhaloIndices
Type: boolean 1
Default value: .true.
Description: Specifies whether subhalo indices should be preset when reading merger trees from a file.
Defined in: function:readConstructorParameters
File: merger_trees.construct.read.F90
Used by:

Name: presetSubhaloMasses
Type: boolean 1
Default value: .true.
Description: Specifies whether subhalo mass should be preset when reading merger trees from a file.
Defined in: function:readConstructorParameters

File: merger_trees.construct.read.F90
Used by:

Name: presetUnphysicalSpins
Type: boolean 1
Default value: .false.
Description: When reading merger trees from file and presetting halo spins, detect unphysical (<=0) spins and preset them using the selected halo spin method.
Defined in: function:readConstructorParameters
File: merger_trees.construct.read.F90
Used by:

Name: pressureCharacteristic
Type: real 1
Default value: 4.54 [Blitz and Rosolowsky, 2006]
Description: The characteristic pressure (given as P_0/k_B in units of K cm^{-3}) in the scaling relation of molecular hydrogen fraction with disk pressure in the "Blitz-Rosolowsky2006" star formation timescale calculation.
Defined in: function:blitz2006ConstructorParameters
File: star_formation.rate_surface_density.disks.Blitz2006.F90
Used by:

Name: pressureExponent
Type: real 1
Default value: 0.92 [Blitz and Rosolowsky, 2006]
Description: The exponent in the scaling relation of molecular hydrogen fraction with disk pressure in the "Blitz-Rosolowsky2006" star formation timescale calculation.
Defined in: function:blitz2006ConstructorParameters
File: star_formation.rate_surface_density.disks.Blitz2006.F90
Used by:

Name: primaryProgenitorDepth
Type: integer 1
Default value: 2
Description: The depth in progenitor ranking for which to store ranked progenitor mass functions. For example, a value of 2 means store mass functions for the most massive, and second most massive progenitor.
Defined in: function:conditionalMFConstructorParameters
File: merger_trees.operators.conditional_mass_function.F90
Used by:

Name: probabilitySuccess
Type: real 1
Description: The probability of success for a single trial.
Defined in: function:negativeBinomialConstructorParameters
File: statistics.distributions.discrete.negative_binomial.F90
Used by:

Name: processDescending

Type: boolean 1
Default value: .true.
Description: If true, causes merger trees to be processed in order of decreasing mass.
Defined in: function:buildConstructorParameters
File: merger_trees.construct.build.F90
Used by:

Name: profileOdeEvolver
Type: boolean 1
Default value: .false.
Description: Specifies whether or not to profile the ODE evolver.
Defined in: function:standardConstructorParameters
File: merger_trees.node_evolver.standard.F90
Used by:

Name: progenitorRedshifts
Type: real 1..
Default value: [1.0d0]
Description: The set of progenitor halo redshifts to use when constructing conditional halo mass functions.
Defined in: function:conditionalMFConstructorParameters
File: merger_trees.operators.conditional_mass_function.F90
Used by:

Name: propertyComment
Type: string 0..1
Description: A descriptive comment for the property variable.
Defined in: function:volumeFunction1DConstructorParameters
File: galacticus.output.analyses.volume_function_1d.F90
Used by:

Name: propertyLabel
Type: string 0..1
Description: A label for the property variable.
Defined in: function:volumeFunction1DConstructorParameters
File: galacticus.output.analyses.volume_function_1d.F90
Used by:

Name: propertyUnits
Type: string 0..1
Description: A human-readable description of the units for the property.
Defined in: function:volumeFunction1DConstructorParameters
File: galacticus.output.analyses.volume_function_1d.F90
Used by:

Name: propertyUnitsInSI
Type: string 0..1
Description: A units for the property in the SI system.
Defined in: function:volumeFunction1DConstructorParameters

File: galacticus.output.analyses.volume_function_1d.F90
Used by:

Name: proposalSize
Type: real 1
Description: The proposal size, γ.
Defined in: function:fixedConstructorParameters
File: posterior_sampling.differential_proposal_size.fixed.F90
Used by:

Name: q
Type: real 0..1
Default value: 1.795 [Comparat et al., 2017]
Description: The parameter $\bar{q}$ in the Bhattacharya et al. [2011] halo mass function fit.
Defined in: function:bhattacharya2011ConstructorParameters
File: structure_formation.halo_mass_function.Bhattacharya2011.F90
Used by:

Name: radii
Type: real 1..*
Description: A list of radii at which to output the mass profile.
Defined in: function:massProfileConstructorParameters
File: nodes.property_extractor.mass_profile.F90
Used by:

Name: radiiRatio
Type: real 1
Default value: 1.0
Description: The parameter β in the relation $r_\mathrm{s} = \beta r_\mathrm{th}$ between k-space sharp and top-hat window function radii in the hybrid window function used for computing the variance in the power spectrum.
Defined in: function:topHatSharpKHybridConstructorParameters
File: structure_formation.power_spectrum.variance.window_function.top_hat_kspace_sharp_-hybrid.F90
Used by:

Name: radius
Type: float 0..
Description: Radii at which the rotation curve should be computed.
Defined in: function:rotationCurveConstructorParameters
File: nBody.operator.rotation_curve.F90
Used by:

Name: radiusEnvironment
Type: real 0..1
Default value: 7.0
Description: The radius of the sphere used to determine the variance in the environmental density.
Defined in: function:normalConstructorParameters
File: structure_formation.halo_environment.normal.F90
Used by:

Name: `radiusFixed`
Type: string 1
Default value: var_str('virial')
Description: The radius to use in the "fixed" galactic structure radius solver algorithm. Allowed options are "virial" and "turnaround".
Defined in: `function:fixedConstructorParameters`
File: `galactic_structure.radius_solver.fixed.F90`
Used by:

Name: `radiusFractionalDecay`
Type: float 1
Default value: 1.0
Description: The truncation scale (in units of the virial radius).
Defined in: `function:truncatedExponentialConstructorParameters`
File: `dark_matter_profiles_DMO.truncated.exponential.F90`
Used by:

Name: `radiusFractionalTruncateMaximum`
Type: float 1
Default value: 4.0
Description: The maximum radius (in units of the virial radius) to finish truncating the density profile.
Defined in: `function:truncatedConstructorParameters`
File: `dark_matter_profiles_DMO.truncated.F90`
Used by:

Name: `radiusFractionalTruncateMinimum`
Type: float 1
Default value: 2.0
Description: The minimum radius (in units of the virial radius) to begin truncating the density profile.
Defined in: `function:truncatedConstructorParameters`
File: `dark_matter_profiles_DMO.truncated.F90`
Used by:

Name: `radiusHalfMass`
Type: real 1
Default value: 1.0
Description: The half mass radius of the Sérsic profile.
Defined in: `function:sersicConstructorParameters`
File: `mass_distributions.spherical.Sersic.F90`
Used by:

Name: `radiusInner`
Type: float 0..
Description: Inner radii of spherical shells within which the velocity dispersion should be computed.
Defined in: `function:velocityDispersionConstructorParameters`
File: `nBody.operator.velocity_dispersion.F90`
Used by:

Name: radiusOuter
Type: float 0..
Description: Outer radii of spherical shells within which the velocity dispersion should be computed.
Defined in: function:velocityDispersionConstructorParameters
File: nBody.operator.velocity_dispersion.F90
Used by:

Name: radiusShock
Type: real 1
Default value: 1.0
Description: The shock radius, s, (in units of the halo virial radius) in the Patej and Loeb [2015] hot halo gas mass distribution model.
Defined in: function:patejLoeb2015ConstructorParameters
File: hot_halo.mass_distribution.PatejLoeb2015.F90
Used by:

Name: radiusSpecifiers
Type: real 1..*
Description: A list of radius specifiers at which to output the velocity dispersion.
Defined in: function:velocityDispersionConstructorParameters
File: nodes.property_extractor.velocity_dispersion.F90
Used by:

Name: radiusSphere
Type: real 0..1
Description: The radius of the sphere within which to measure environmental overdensity.
Defined in: function:environmentalOverdensityConstructorParameters
File: nBody.operator.environmental_overdensity.F90
Used by:

Name: radiusTree
Type: float 1
Default value: spread(-1.0d0,1,fixedHalosCount)
Description: Specifies the radii within which halo masses are specified when building halos.
Defined in: function:fixedMassConstructorParameters
File: merger_trees.construct.build.masses.fixed_mass.F90
Used by:

Name: radiusTruncateOverRadiusVirial
Type: real 0..1
Description: Radius (in units of the virial radius) at which to truncate halo profiles.
Defined in: function:particulateConstructorParameters
File: merger_trees.operators.particulate.F90
Used by:

Name: randomErrorMaximum
Type: float 0..1
Default value: 0.07
Description: The minimum random error for stellar masses.

Defined in: function:morphologicalFractionGAMAMoffett2016ConstructorParameters
File: galacticus.output.analyses.morphological_fraction.GAMA_Moffett2016.F90
Used by:

Name: randomErrorMinimum
Type: float 0..1
Default value: 0.07
Description: The minimum random error for stellar masses.
Defined in: function:morphologicalFractionGAMAMoffett2016ConstructorParameters
File: galacticus.output.analyses.morphological_fraction.GAMA_Moffett2016.F90
Used by:

Name: randomErrorPolynomialCoefficient
Type: float 0..1
Default value: [0.0d0]
Description: The coefficients of the random error polynomial.
Defined in: function:morphologicalFractionGAMAMoffett2016ConstructorParameters
File: galacticus.output.analyses.morphological_fraction.GAMA_Moffett2016.F90
Used by:

Name: randomSampleCount
Type: float 1
Default value: var_str('*10')
Description: The number of random points to use when constructing random catalogs. Can be either a fixed number or, if prefixed with "*", a multiplicative factor.
Defined in: function:catalogProjectedCorrelationFunctionConstructorParameters
File: tasks.catalog_projected_correlation_function.F90
Used by:

Name: randomSeed
Type: integer 1
Default value: 219
Description: A seed value for the random number generator.
Defined in: subroutine:pseudoRandomInitialize
File: numerical.random.F90
Used by:

Name: randomSpinResetMassFactor
Type: double 1
Default value: 2.0
Description: The factor by which a node must increase in mass before its spin parameter is reset.
Defined in: subroutine:Node_Component_Spin_Random_Initialize
File: objects.nodes.components.spin.random.F90
Used by:

Name: randomize
Type: boolean 1
Default value: .false.
Description: If true, randomize models (i.e. change the random seed).

Defined in: function:galaxyPopulationConstructorParameters
File: models.likelihoods.galaxy_population.F90
Used by:

Name: rangeLower
Type: float 0..1
Description: Lower integration limit for the normal distribution weight operator.
Defined in: function:normalConstructorParameters
File: galacticus.output.analyses.property_operator.normal.F90
Used by:

Name: rangeUpper
Type: float 0..1
Description: Upper integration limit for the normal distribution weight operator.
Defined in: function:normalConstructorParameters
File: galacticus.output.analyses.property_operator.normal.F90
Used by:

Name: rate
Type: real 1
Description: The rate parameter of the negative exponential distribution function.
Defined in: function:negativeExponentialConstructorParameters
File: statistics.distributions.negative_exponential.F90
Used by:

Name: rateAdjust
Type: real 1
Default value: 0.3
Description: The dimensionless multiplier for the rate at which the halo gas content adjusts to changes in the filtering mass.
Defined in: function:naozBarkana2007ConstructorParameters
File: accretion.halo.Naoz_Barkana_2007.F90
Used by:

Name: rateFractionalMaximum
Type: string 1
Default value: 10.0
Description: The maximum fractional mass loss rate per dynamical time in the simple model of mass loss from spheroids due to tidal stripping.
Defined in: function:simpleConstructorParameters
File: tidal_stripping.mass_loss_rate.spheroids.simple.F90
Used by:

Name: ratioEarlyType
Type: real 0..1
Default value: 0.5
Description: The minimum spheroid-to-total ratio for a galaxy to be classified as "early-type" when constructing the Galaxy and Mass Assembly (GAMA) early-type fraction function.
Defined in: function:morphologicalFractionGAMAMoffett2016ConstructorParameters

File: galacticus.output.analyses.morphological_fraction.GAMA_Moffett2016.F90
Used by:

Name: ratioEarlyTypeError
Type: real 0..1
Default value: 0.3
Description: The error in spheroid fraction to be used when constructing the GAMA early-type fraction function.
Defined in: function:morphologicalFractionGAMAMoffett2016ConstructorParameters
File: galacticus.output.analyses.morphological_fraction.GAMA_Moffett2016.F90
Used by:

Name: ratioMassBurst
Type: real 1
Default value: 0.05
Description: The mass ratio above which mergers are considered to trigger a burst.
Defined in: function:baugh2005ConstructorParameters
File: satellites.merging.mass_movements.Baugh2005.F90
Used by:

Name: realizationCount
Type: integer 1
Description: The number of realizations of the stochastic likelihood to compute at unit temperature.
Defined in: function:multivariateNormalStochasticConstructorParameters
File: models.likelihoods.multivariate_normal.stochastic.F90
Used by:

Name: realizationCountMinimum
Type: integer 1
Description: The minimum number of realizations of the stochastic likelihood to compute at higher temperatures.
Defined in: function:multivariateNormalStochasticConstructorParameters
File: models.likelihoods.multivariate_normal.stochastic.F90
Used by:

Name: recycledFraction
Type: real 1
Default value: 0.0
Description: The recycled fraction to use in the instantaneous stellar evolution approximation. (If not specified it will be computed internally.)
Defined in: function:standardConstructorParameters
File: stellar_populations.standard.F90
Used by:

Name: redshift
Type: real 1
Default value: 0.0
Description: The redshift at which the transfer function is defined.
Defined in: function:identityConstructorParameters

File: structure_formation.transfer_function.identity.F90
Used by:

Name: redshiftBand
Type: float 0..1
Description: The redshift of the band (if not the output redshift).
Defined in: function:lmnstyStllrChrltFll2000ConstructorParameters
File: nodes.property_extractor.luminosity_stellar.dust.CharlotFall2000.F90
Used by:

Name: redshiftBase
Type: real 1
Default value: 0.0
Description: The redshift at which to plant the base node when building the smoothly accreting merger tree.
Defined in: function:smoothAccretionConstructorParameters
File: merger_trees.construct.smooth_accretion.F90
Used by:

Name: redshiftBin
Type: integer 1
Description: The redshift bin (0, 1, 2, 3, 4, 5, 6, or 7) of the Tomczak et al. [2014] mass function to use.
Defined in: function:tomczak2014ZFOURGEConstructorParameters
File: geometry.surveys.Tomczak-2014-ZFOURGE.F90
Used by:

Name: redshiftCutOff
Type: real 1
Default value: 0.0
Description: The redshift below which cooling is suppressed in the "cut-off" cooling rate modifier method.
Defined in: function:cutOffConstructorParameters
File: cooling.cooling_rate.cut_off.F90
Used by:

Name: redshiftEarliest
Type: real 1
Default value: 0.0
Description: Redshift at which to truncate merger tree branches.
Defined in: function:pruneByTimeConstructorParameters
File: merger_trees.operators.prune_by_time.F90
Used by:

Name: redshiftExponent
Type: real 1
Default value: -1.5d0
Description: The exponent of redshift in the velocity maximum scaling model for outflow reincorporation.
Defined in: function:velocityMaximumScalingConstructorParameters

File: hot_halo.outflow_reincorporation.velocity_maximum_scaling.F90
Used by:

Name: redshiftInterval
Type: integer 0..1
Description: The redshift interval (1, 2, or 3) to use.
Defined in: function:stellarVsHaloMassRelationLeauthaud2012ConstructorParameters
File: galacticus.output.analyses.stellar_vs_halo_mass_relation.COSMOS_Leauthaud2012.F90
Used by:

Name: redshiftMaximum
Type: real 1
Default value: 10.0
Description: The maximum redshift at which to tabulate excursion set solutions.
Defined in: function:excursionSetsConstructorParameters
File: tasks.excursion_sets.F90
Used by:

Name: redshiftMinimum
Type: real 1
Default value: 0.0
Description: The minimum redshift at which to tabulate excursion set solutions.
Defined in: function:excursionSetsConstructorParameters
File: tasks.excursion_sets.F90
Used by:

Name: redshiftReionization
Type: real 1
Default value: 9.97 (Hinshaw et al. 2012; CMB+H_0+BAO)
Description: The redshift below which baryonic accretion is suppressed.
Defined in: function:simpleConstructorParameters
File: accretion.halo.simple.F90
Used by:

Name: redshifts
Type: real 1..*
Default value: [0.0d0]
Description: A list of (space-separated) redshifts at which Galacticus results should be output. Redshifts need not be in any particular order.
Defined in: function:listConstructorParameters
File: output.times.list.F90
Used by:

Name: regridCount
Type: integer 1
Default value: 100
Description: Number of points in time to use when regridding merger trees.
Defined in: function:regridTimesConstructorParameters
File: merger_trees.operators.regrid_times.F90

Used by:

Name: `reionizationRedshift`
Type: real 1
Default value: 9.97 (Hinshaw et al. 2012; CMB+H_0+BAO)
Description: The redshift of reionization in the simple IGM state model.
Defined in: `function:simpleIGMConstructorParameters`
File: `intergalactic_medium.state.simple.F90`
Used by:

Name: `reionizationTemperature`
Type: real 1
Default value: 1.0×10^4
Description: The post-reionization temperature (in units of Kelvin) in the simple IGM state model.
Defined in: `function:simpleIGMConstructorParameters`
File: `intergalactic_medium.state.simple.F90`
Used by:

Name: `reportCount`
Type: integer 1
Default value: 10
Description: The number of steps between issuing reports.
Defined in: `function:particleSwarmConstructorParameters`
File: `posterior_sampling.simulation.particle_swarm.F90`
Used by:

Name: `rescaleMaximum`
Type: integer 0..
Default value: 20
Description: The maximum allowed number of tolerance rescalings.
Defined in: `function:augmentConstructorParameters`
File: `merger_trees.operators.augment.F90`
Used by:

Name: `restoreLevels`
Type: boolean 1
Default value: .true.
Description: If true the level reached by each chain is restored on restarts. Otherwise, the level is initialized to zero (which may be useful for stochastic likelihoods).
Defined in: `function:independentLikelihoodsSequentialConstructorParameters`
File: `models.likelihoods.independent_likelihoods.sequential.F90`
Used by:

Name: `restoreState`
Type: logical 1
Description: If true, restore the state of the simulation.
Defined in: `function:resumeConstructorParameters`
File: `posterior_sampling.state.initialize.resume.F90`
Used by:

Name: `resume`
Type: integer 1
Default value: .false.
Description: If true, resume from a previous set of log files.
Defined in: `function:particleSwarmConstructorParameters`
File: `posterior_sampling.simulation.particle_swarm.F90`
Used by:

Name: `retryMaximum`
Type: integer 0..
Default value: 50
Description: The number of tree build attempts to complete before rescaling the tolerance.
Defined in: `function:augmentConstructorParameters`
File: `merger_trees.operators.augment.F90`
Used by:

Name: `reweightTrees`
Type: boolean 1
Default value: .false.
Description: Specifies whether merger tree weights should be recomputed from the halo mass function.
Defined in: `function:galacticusConstructorParameters`
File: `merger_trees.construct.read.importer.galacticus.F90`
Used by:

Name: `rootVariance`
Type: float 0..1
Description: Root variance for the normal distribution weight operator.
Defined in: `function:normalConstructorParameters`
File: `galacticus.output.analyses.property_operator.normal.F90`
Used by:

Name: `running`
Type: real 0..1
Default value: 0.0
Description: The running, $\mathrm{d}n_\mathrm{s}/\mathrm{d}\ln k$, of the power spectrum index.
Defined in: `function:powerLawConstructorParameters`
File: `structure_formation.power_spectrum.primordial.power_law.F90`
Used by:

Name: `sample`
Type: string 1
Default value: var_str('SMCbar')
Description: The name of the sample from Gordon et al. [2003] to use in dust attenuation calculations.
Defined in: `function:gordon2003ConstructorParameters`
File: `stellar_populations.spectra.dust_attenuation.Gordon2003.F90`
Used by:

Name: `sampleOutliers`

Type: integer 1
Default value: .true.
Description: If true, sample from outlier states.
Defined in: function:differentialEvolutionConstructorParameters
File: posterior_sampling.simulation.differential_evolution.F90
Used by:

Name: sampleParticleNumber
Type: boolean 0..1
Default value: .false.
Description: If true, the number of particles in each halo will be sampled from a Poisson distribution with the expected mean. Otherwise, the number is set equal to that expectation.
Defined in: function:particulateConstructorParameters
File: merger_trees.operators.particulate.F90
Used by:

Name: sampleRate
Type: integer 0..1
Default value: 1_c_size_t
Description: One in [sampleRate] particles will be sampled when computed environmental overdensities.
Defined in: function:environmentalOverdensityConstructorParameters
File: nBody.operator.environmental_overdensity.F90
Used by:

Name: satelliteBoundMassInitializeType
Type: string 1
Default value: var_str('basicMass')
Description: Specify how to initialize the bound mass of a satellite halo. By default, the initial bound mass of a satellite halo is set to the node mass.
Defined in: subroutine:Node_Component_Satellite_Orbiting_Initialize
File: objects.nodes.components.satellite.orbiting.F90
Used by:

Name: satelliteBoundMassIsInactive
Type: boolean 1
Default value: .false.
Description: Specifies whether or not the bound mass variable of the standard satellite component is inactive (i.e. does not appear in any ODE being solved).
Defined in: subroutine:Node_Component_Satellite_Standard_Initialize
File: objects.nodes.components.satellite.standard.F90
Used by:

Name: satelliteDensityContrast
Type: double 1
Default value: 200.0
Description: The density contrast of the satellite halo. If [satelliteBoundMassInitializeType] is set to 'densityContrast', this value will be used to compute the initial bound mass of the satellite halo.
Defined in: subroutine:Node_Component_Satellite_Orbiting_Initialize

File: objects.nodes.components.satellite.orbiting.F90
Used by:

Name: satelliteMaximumRadiusOverVirialRadius
Type: double 1
Default value: 1.0
Description: The maximum radius of the satellite halo in units of its virial radius. If [satelliteBoundMassInitializeType] is set to 'maximumRadius', this value will be used to compute the initial bound mass of the satellite halo assuming that its density profile is 0 beyond this maximum radius.
Defined in: subroutine:Node_Component_Satellite_Orbiting_Initialize
File: objects.nodes.components.satellite.orbiting.F90
Used by:

Name: satelliteOffset
Type: boolean 0..1
Default value: .true.
Description: If true, offset particle representations to the positions/velocities of satellites.
Defined in: function:particulateConstructorParameters
File: merger_trees.operators.particulate.F90
Used by:

Name: satelliteOrbitResetOnHaloFormation
Type: boolean 1
Default value: .false.
Description: Specifies whether satellite virial orbital parameters should be reset on halo formation events.
Defined in: subroutine:Node_Component_Satellite_Standard_Initialize
File: objects.nodes.components.satellite.standard.F90
Used by:

Name: satelliteOrbitStoreOrbitalParameters
Type: boolean 1
Default value: .true.
Description: Specifies whether satellite virial orbital parameters should be stored (otherwise they are computed again—possibly at random—each time they are requested).
Defined in: subroutine:Node_Component_Satellite_Standard_Initialize
File: objects.nodes.components.satellite.standard.F90
Used by:

Name: satelliteOrbitingDestructionMass
Type: double 1
Default value: 0.01
Description: The mass (possibly fractional—see [satelliteOrbitingDestructionMassIsFractional]) below which the satellite is considered to be tidally destroyed and merged with the central halo.
Defined in: subroutine:Node_Component_Satellite_Orbiting_Initialize
File: objects.nodes.components.satellite.orbiting.F90
Used by:

Name: satelliteOrbitingDestructionMassIsFractional

Type: double 1
Default value: .true.
Description: If true, then `[satelliteOrbitingDestructionMass]` specifies the fractional mass a halo must reach before it is tidally destroyed. Otherwise, `[satelliteOrbitingDestructionMass]` specifies an absolute mass.
Defined in: subroutine:Node_Component_Satellite_Orbiting_Initialize
File: objects.nodes.components.satellite.orbiting.F90
Used by:

Name: `satelliteOutputVirialOrbit`
Type: string 1
Default value: .false.
Description: Specifies whether the
normalfont
ttfamily virialOrbit method of the
normalfont
ttfamily standard implemention of the
normalfont
ttfamily component class should be output.
Defined in: `subroutine:nodeClassHierarchyInitialize`
File: `objects.nodes.components.Inc`
Used by:

Name: `scale`
Type: real 1
Description: The scale parameter of the Cauchy distribution function.
Defined in: function:cauchyConstructorParameters
File: statistics.distributions.Cauchy.F90
Used by:

Name: `scaleADAFRadiativeEfficiency`
Type: real 1
Default value: .true.
Description: Specifies whether the radiative efficiency of the ADAF component in a switched accretion disk scales with accretion rate.
Defined in: function:switchedConstructorParameters
File: accretion_disks.switched.F90
Used by:

Name: `scaleFactorExponent`
Type: integer 1
Description: The exponent of the cosmological scale factor needed to convert the velocities to physical units.
Defined in: function:mergerTreeFileBuilderConstructorParameters
File: tasks.merger_tree_file_builder.F90
Used by:

Name: `scaleHeight`
Type: real 1

Default value: 0.137 [Kregel et al., 2002]
Description: The scale height of the exponential disk profile.
Defined in: function:exponentialDiskConstructorParameters
File: mass_distributions.cylindrical.exponential_disk.F90
Used by:

Name: scaleLength
Type: real 1
Default value: 1.0
Description: The scale radius of the NFW profile.
Defined in: function:nfwConstructorParameters
File: mass_distributions.spherical.NFW.F90
Used by:

Name: scaleNodesByLogMass
Type: boolean 1
Default value: .true.
Description: Specifies whether or not node sizes should be scaled by the logarithm of their mass.
Defined in: function:dumpToGraphVizConstructorParameters
File: merger_trees.operators.dump_to_GraphViz.F90
Used by:

Name: scaleRadiiFailureIsFatal
Type: boolean 1
Default value: .true.
Description: Specifies whether failure to set a node scale radii should be regarded as a fatal error. (If not, a fallback method to set scale radius is used in such cases.)
Defined in: function:readConstructorParameters
File: merger_trees.construct.read.F90
Used by:

Name: scaleRadius
Type: real 1
Default value: 1.0
Description: The scale radius of the exponential disk profile.
Defined in: function:exponentialDiskConstructorParameters
File: mass_distributions.cylindrical.exponential_disk.F90
Used by:

Name: scatter
Type: string 1
Description: Scatter in the mass function incompleteness surface brightness model.
Defined in: function:surfaceBrightnessConstructorParameters
File: statistics.mass_function.incompleteness.surface_brightness.F90
Used by:

Name: scatterComment
Type: string 0..1
Description: A descriptive comment for the scatter.

Defined in: function:scatterFunction1DConstructorParameters
File: galacticus.output.analyses.scatter_function_1d.F90
Used by:

Name: scatterCovarianceTarget
Type: real 0..1
Description: The target function covariance for likelihood calculations.
Defined in: function:scatterFunction1DConstructorParameters
File: galacticus.output.analyses.scatter_function_1d.F90
Used by:

Name: scatterLabel
Type: string 0..1
Description: A label for the scatter.
Defined in: function:scatterFunction1DConstructorParameters
File: galacticus.output.analyses.scatter_function_1d.F90
Used by:

Name: scatterUnits
Type: string 0..1
Description: A human-readable description of the units for the scatter.
Defined in: function:scatterFunction1DConstructorParameters
File: galacticus.output.analyses.scatter_function_1d.F90
Used by:

Name: scatterUnitsInSI
Type: string 0..1
Description: A units for the scatter in the SI system.
Defined in: function:scatterFunction1DConstructorParameters
File: galacticus.output.analyses.scatter_function_1d.F90
Used by:

Name: scatterValueTarget
Type: real 0..1
Description: The target function for likelihood calculations.
Defined in: function:scatterFunction1DConstructorParameters
File: galacticus.output.analyses.scatter_function_1d.F90
Used by:

Name: selection
Type: string 0..1
Default value: var_str('all')
Description: Selects the type of halo to ouput. Allowed options are "all", "hosts", and "satellites".
Defined in: function:particulateConstructorParameters
File: merger_trees.operators.particulate.F90
Used by:

Name: selfBoundParticlesOnly
Type: logical 0..1

Description: If true, the velocity dispersion is computed only for self-bound particles.
Defined in: function:velocityDispersionConstructorParameters
File: nBody.operator.velocity_dispersion.F90
Used by:

Name: separationCount
Type: float 1
Default value: 15
Description: The number of bins in separation to compute in a mock catalog correlation function calculation.
Defined in: function:catalogProjectedCorrelationFunctionConstructorParameters
File: tasks.catalog_projected_correlation_function.F90
Used by:

Name: separationMaximum
Type: string 1
Description: The maximum separation at which to compute the projected correlation function.
Defined in: function:haloModelProjectedCorrelationFunctionConstructorParameters
File: tasks.halo_model.projected_correlation_function.F90
Used by:

Name: separationMinimum
Type: string 1
Description: The minimum separation at which to compute the projected correlation function.
Defined in: function:haloModelProjectedCorrelationFunctionConstructorParameters
File: tasks.halo_model.projected_correlation_function.F90
Used by:

Name: separationRadialMaximum
Type: float 1
Default value: 40.0
Description: The maximum radial separation of galaxies to consider when computing projected correlation functions.
Defined in: function:catalogProjectedCorrelationFunctionConstructorParameters
File: tasks.catalog_projected_correlation_function.F90
Used by:

Name: separations
Type: float 0..1
Description: The separations corresponding to bin centers.
Defined in: function:correlationFunctionConstructorParameters
File: galacticus.output.analyses.correlation_function.F90
Used by:

Name: shape
Type: real 1
Description: The shape parameter of the gamma distribution function.
Defined in: function:gammaConstructorParameters
File: statistics.distributions.Gamma.F90

Used by:

Name: `sigma`
Type: real 1
Default value: 0.69
Description: The width of the lognormal part of the Chabrier [2001] IMF.
Defined in: `function:chabrier2001ConstructorParameters`
File: `stellar_populations.initial_mass_functions.Chabrier2001.F90`
Used by:

Name: `sigmaBuffer`
Type: real 1
Default value: 3.0
Description: The buffer size in units of the likelihood error to use when deciding whether to emulate the likelihood.
Defined in: `function:gaussianRegressionConstructorParameters`
File: `models.likelihoods.gaussian_regression.F90`
Used by:

Name: `sigmaLogMstar`
Type: real 1
Default value: 0.206 (Leauthaud et al. 2011; z_1 sample using their `SIG_MOD1` method)
Description: The parameter $\sigma_{\log M_\star}$ from the fitting functions of Behroozi et al. [2010].
Defined in: `function:behroozi2010ConstructorParameters`
File: `halo_model.conditional_mass_function.Behroozi2010.F90`
Used by:

Name: `sigmaRatioHigh`
Type: real 1
Default value: [+0.091d0,+0.139d0,+0.187d0]
Description: Values of the σ parameter of the Jiang et al. [2014] orbital velocity distribution for the three host halo mass ranges, and the 0.05–0.5 mass ratio.
Defined in: `function:jiang2014ConstructorParameters`
File: `satellites.merging.virial_orbits.Jiang2014.F90`
Used by:

Name: `sigmaRatioIntermediate`
Type: real 1
Default value: [+0.073d0,+0.083d0,+0.118d0]
Description: Values of the σ parameter of the Jiang et al. [2014] orbital velocity distribution for the three host halo mass ranges, and the 0.005–0.05 mass ratio.
Defined in: `function:jiang2014ConstructorParameters`
File: `satellites.merging.virial_orbits.Jiang2014.F90`
Used by:

Name: `sigmaRatioLow`
Type: real 1
Default value: [+0.077d0,+0.094d0,+0.072d0]
Description: Values of the σ parameter of the Jiang et al. [2014] orbital velocity distribution for the

three host halo mass ranges, and the 0.0001–0.005 mass ratio.
Defined in: function:jiang2014ConstructorParameters
File: satellites.merging.virial_orbits.Jiang2014.F90
Used by:

Name: sigma_8
Type: real 1
Default value: 0.8111 (Planck Collaboration et al. 2018; TT,TE,EE+lowE+lensing)
Description: The fractional mass fluctuation in the linear density field at the present day in spheres of radius 8 Mpc/h.
Defined in: function:filteredPowerConstructorParameters
File: structure_formation.cosmological_mass_variance.filtered_power_spectrum.F90
Used by:

Name: sizeGridFFT
Type: integer 1
Default value: 64
Description: The size of the FFT grid to use in computing window functions for mass function covariance matrices.
Defined in: function:massFunctionCovarianceConstructorParameters
File: tasks.mass_function_covariance.F90
Used by:

Name: sizeSource
Type: real 0..1
Default value: 0.001
Description: The source size to assume for gravitational lensing calculations.
Defined in: function:grvtnlLnsngConstructorParameters
File: galacticus.output.analyses.distribution_operator.gravitational_lensing.F90
Used by:

Name: sizeSourceLensing
Type: float 0..1
Default value: 2.0×10^{-3}
Description: The characteristic source size for gravitational lensing calculations.
Defined in: function:massFunctionStellarZFOURGEConstructorParameters
File: galacticus.output.analyses.mass_function_stellar.ZFOURGE.F90
Used by:

Name: slope
Type: string 1
Description: Slope of mass function incompleteness surface brightness model, i.e. α in $\mu(M) = \alpha \log_{10}(M/M_0) + \beta$.
Defined in: function:surfaceBrightnessConstructorParameters
File: statistics.mass_function.incompleteness.surface_brightness.F90
Used by:

Name: smoothAccretion
Type: boolean 1

Default value: .true.
Description: Specifies whether or not to include smooth accretion in subresolution accretion rates when constructing merger trees using the generalized Press-Schechter branching algorithm.
Defined in: function:generalizedPressSchechterConstructorParameters
File: merger_trees.branching_probability.generalized_Press_Schechter.F90
Used by:

Name: snapTolerance
Type: real 1
Default value: 0.0
Description: The fractional tolerance used in deciding if a node should be snapped to a time on the grid.
Defined in: function:regridTimesConstructorParameters
File: merger_trees.operators.regrid_times.F90
Used by:

Name: snapshotRedshifts
Type: real 0..
Description: The redshifts at which merger trees are to regridded when the [snapshotSpacing]=list option is selected.
Defined in: function:regridTimesConstructorParameters
File: merger_trees.operators.regrid_times.F90
Used by:

Name: snapshotSpacing
Type: string 1
Default value: var_str('logarithmic')
Description: Type of spacing to use in merger tree regridding (linear or logarithmic).
Defined in: function:regridTimesConstructorParameters
File: merger_trees.operators.regrid_times.F90
Used by:

Name: solutionTolerance
Type: real 1
Default value: 1.0×10^{-2}
Description: Maximum allowed mean fractional error in the radii of all components when seeking equilibrium solutions for galactic structure.
Defined in: function:equilibriumConstructorParameters
File: galactic_structure.radius_solver.equilibrium.F90
Used by:

Name: solveForInactiveProperties
Type: boolean 1
Default value: .true.
Description: If true, galactic structure is solved for during evaluation of inactive property integrals. Otherwise, structure is not solved for during this phase—this should only be used if the inactive property integrands *do not* depend on galactic structure.
Defined in: function:equilibriumConstructorParameters
File: galactic_structure.radius_solver.equilibrium.F90

Used by:

Name: `sourceAngularMomentumSpecificMean`
Type: string 1
Default value: var_str('hotGas')
Description: The component ("`hotGas`" or "`darkMatter`") from which the mean specific angular momentum should be computed for calculations of cooling gas specific angular momentum.
Defined in: function:constantRotationConstructorParameters
File: cooling.specific_angular_momentum.constant_rotation.F90
Used by:

Name: `sourceNormalizationRotation`
Type: string 1
Default value: var_str('hotGas')
Description: The component ("`hotGas`" or "`darkMatter`") from which the constant rotation speed should be computed for calculations of cooling gas specific angular momentum.
Defined in: function:constantRotationConstructorParameters
File: cooling.specific_angular_momentum.constant_rotation.F90
Used by:

Name: `spheroidAngularMomentumAtScaleRadius`
Type: double 1
Default value: spheroidAngularMomentumAtScaleRadiusDefault (I_2/I_3 where $I_n = \int_0^\infty \rho(r) r^n \mathrm{d}r$, where $\rho(r)$ is the spheroid density profile, unless either I_2 or I_3 is infinite, in which case a default of $1/2$ is used instead.)
Description: The assumed ratio of the specific angular momentum at the scale radius to the mean specific angular momentum of the standard spheroid component.
Defined in: subroutine:Node_Component_Spheroid_Standard_Thread_Initialize
File: objects.nodes.components.spheroid.standard.F90
Used by:

Name: `spheroidEnergeticOutflowMassRate`
Type: double 1
Default value: 1.0×10^{-2}
Description: The proportionallity factor relating mass outflow rate from the spheroid to the energy input rate divided by $V^2_{\rm spheroid}$.
Defined in: subroutine:Node_Component_Spheroid_Standard_Initialize
File: objects.nodes.components.spheroid.standard.F90
Used by:

Name: `spheroidLuminositiesStellarInactive`
Type: boolean 1
Default value: .false.
Description: Specifies whether or not spheroid stellar luminosities are inactive properties (i.e. do not appear in any ODE being solved).
Defined in: subroutine:Node_Component_Spheroid_Standard_Initialize
File: objects.nodes.components.spheroid.standard.F90
Used by:

Name: spheroidMassToleranceAbsolute
Type: double 1
Default value: 1.0×10^{-6}
Description: The mass tolerance used to judge whether the spheroid is physically plausible.
Defined in: subroutine:Node_Component_Spheroid_Standard_Initialize
File: objects.nodes.components.spheroid.standard.F90
Used by:

Name: spheroidOutflowTimescaleMinimum
Type: double 1
Default value: 1.0×10^{-3}
Description: The minimum timescale (in units of the spheroid dynamical time) on which outflows may deplete gas in the spheroid.
Defined in: subroutine:Node_Component_Spheroid_Standard_Initialize
File: objects.nodes.components.spheroid.standard.F90
Used by:

Name: spheroidOutputStarFormationRate
Type: string 1
Default value: .false.
Description: Specifies whether the
normalfont
ttfamily starFormationRate method of the
normalfont
ttfamily standard implemention of the
normalfont
ttfamily component class should be output.
Defined in: subroutine:nodeClassHierarchyInitialize
File: objects.nodes.components.Inc
Used by:

Name: spheroidRadiusFraction
Type: real 1
Default value: 0.0
Description: The fraction of the spheroid radius at which merging black holes will be initially placed.
Defined in: function:spheroidRadiusFractionConstructorParameters
File: black_holes.binaries.initial_separation.spheroid_size_fraction.F90
Used by:

Name: spheroidStarFormationInSatellites
Type: boolean 1
Default value: .true.
Description: Specifies whether or not star formation occurs in spheroids in satellites.
Defined in: subroutine:Node_Component_Spheroid_Standard_Initialize
File: objects.nodes.components.spheroid.standard.F90
Used by:

Name: spheroidStarFormationTimescaleMinimum
Type: double 1

Default value: 1.0×10^{-3}
Description: The minimum timescale (in units of the halo dynamical time) on which star formation may occur in the spheroid.
Defined in: subroutine:Node_Component_Spheroid_Very_Simple_Initialize
File: objects.nodes.components.spheroid.very_simple.F90
Used by:

Name: spheroidToTotalRatioThreshold
Type: real 0..1
Description: The parameter R_0 appearing in the stellar mass-weight morphology threshold for the stellar mass-weighted morphology galactic filter class.
Defined in: function:stellarMassMorphologyConstructorParameters
File: galactic.filters.stellar_mass_morphology.F90
Used by:

Name: spheroidVerySimpleMassScaleAbsolute
Type: double 1
Default value: 100.0
Description: The absolute mass scale below which calculations in the very simple spheroid component are allowed to become inaccurate.
Defined in: subroutine:Node_Component_Spheroid_Very_Simple_Initialize
File: objects.nodes.components.spheroid.very_simple.F90
Used by:

Name: spheroidVerySimpleTrackAbundances
Type: boolean 0..1
Default value: .false.
Description: Specifies whether or not to track abundances in the very simple spheroid component.
Defined in: subroutine:Node_Component_Spheroid_Very_Simple_Initialize
File: objects.nodes.components.spheroid.very_simple.F90
Used by:

Name: spheroidVerySimpleTrackLuminosities
Type: boolean 0..1
Default value: .false.
Description: Specifies whether or not to track stellar luminosities in the very simple disk component.
Defined in: subroutine:Node_Component_Spheroid_Very_Simple_Initialize
File: objects.nodes.components.spheroid.very_simple.F90
Used by:

Name: spin
Type: real 1
Default value: 0.03687 [Bett et al., 2007]
Description: The fixed value of spin in a δ-function spin distribution.
Defined in: function:deltaFunctionConstructorParameters
File: dark_matter_halos.spins.distributions.delta_function.F90
Used by:

Name: spinMaximum

Type: real 0..1
Default value: 0.5
Description: Maximum spin for which the distribution function should be calculated.
Defined in: function:haloSpinDistributionConstructorParameters
File: tasks.halo_spin_distribution.F90
Used by:

Name: spinMinimum
Type: real 0..1
Default value: 3.0×10^{-4}
Description: Minimum spin for which the distribution function should be calculated.
Defined in: function:haloSpinDistributionConstructorParameters
File: tasks.halo_spin_distribution.F90
Used by:

Name: spinPointsPerDecade
Type: real 0..1
Default value: 10.0
Description: Number of points per decade of spin at which to calculate the distribution.
Defined in: function:haloSpinDistributionConstructorParameters
File: tasks.halo_spin_distribution.F90
Used by:

Name: spinVitvitskaMassExponent
Type: double 1
Default value: 1.0
Description: The exponent of mass ratio appearing in the orbital angular momentum term in the Vitvitska spin model.
Defined in: subroutine:Node_Component_Spin_Vitvitska_Bindings
File: objects.nodes.components.spin.Vitvitska.F90
Used by:

Name: stabilityThresholdGaseous
Type: real 1
Default value: 0.7
Description: The stability threshold in the Efstathiou et al. [1982] algorithm for purely gaseous disks.
Defined in: function:efstathiou1982ConstructorParameters
File: galactic_dynamics.bar_instability.Efstathiou1982.F90
Used by:

Name: stabilityThresholdStellar
Type: real 1
Default value: 1.1
Description: The stability threshold in the Efstathiou et al. [1982] algorithm for purely stellar disks.
Defined in: function:efstathiou1982ConstructorParameters
File: galactic_dynamics.bar_instability.Efstathiou1982.F90
Used by:

Name: starFormationFrequencyNormalization

Type: real 1
Default value: 5.25×10^{-10} [Leroy et al., 2008]
Description: The star formation frequency (in the low-density limit and in units of yr^{-1}) in the "Blitz-Rosolowsky2006" star formation timescale calculation.
Defined in: function:blitz2006ConstructorParameters
File: star_formation.rate_surface_density.disks.Blitz2006.F90
Used by:

Name: startTime
Type: string 1
Description: The definition of start time (absolute time or age).
Defined in: function:sedFitConstructorParameters
File: models.likelihoods.SED_fit.F90
Used by:

Name: starveSatellites
Type: boolean 1
Default value: .false.
Description: Specifies whether or not the hot halo should be removed ("starved") when a node becomes a satellite.
Defined in: subroutine:Node_Component_Hot_Halo_Standard_Initialize
File: objects.nodes.components.hot_halo.standard.F90
Used by:

Name: starveSatellitesOutflowed
Type: boolean 1
Default value: .false.
Description: Specifies whether or not the outflowed hot halo should be removed ("starved") when a node becomes a satellite.
Defined in: subroutine:Node_Component_Hot_Halo_Standard_Initialize
File: objects.nodes.components.hot_halo.standard.F90
Used by:

Name: stateFileRoot
Type: string 1
Default value: var_str('none')
Description: The root name of files to which the internal state is written (to permit restarts).
Defined in: subroutine:State_Initialize
File: galacticus.state.F90
Used by:

Name: stateRetrieveFileRoot
Type: string 1
Default value: var_str('none')
Description: The root name of files to which the internal state is retrieved from (to restart).
Defined in: subroutine:State_Initialize
File: galacticus.state.F90
Used by:

Name: `stateSwapCount`
Type: integer 1
Default value: 10
Description: The number of steps between state swap steps.
Defined in: function:differentialEvolutionConstructorParameters
File: posterior_sampling.simulation.differential_evolution.F90
Used by:

Name: `stellarDensityChangeBinaryMotion`
Type: boolean 1
Default value: .true.
Description: The change in density due to the black hole's motion.
Defined in: function:standardConstructorParameters
File: black_holes.binaries.separation_growth_rate.standard.F90
Used by:

Name: `stellarPopulationLuminosityIntegrationToleranceDegrade`
Type: real 1
Default value: .false.
Description: If `true`, automatically degrade the relative tolerance used when integrating the flux of stellar populations through filters to ensure convergence.
Defined in: subroutine:Stellar_Population_Luminosity_Tabulate
File: stellar_populations.luminosities.F90
Used by:

Name: `stellarPopulationLuminosityIntegrationToleranceRelative`
Type: real 1
Default value: 4.0×10^{-3}
Description: The relative tolerance used when integrating the flux of stellar populations through filters.
Defined in: subroutine:Stellar_Population_Luminosity_Tabulate
File: stellar_populations.luminosities.F90
Used by:

Name: `stellarPopulationLuminosityMaximumAgeExceededIsFatal`
Type: boolean 1
Default value: .true.
Description: Specifies whether or not exceeding the maximum available age of the stellar population is fatal.
Defined in: subroutine:Stellar_Population_Luminosity_Tabulate
File: stellar_populations.luminosities.F90
Used by:

Name: `stellarPopulationLuminosityStoreDirectory`
Type: string 1
Default value: galacticusPath(pathTypeDataDynamic)//'stellarPopulations'
Description: Specifies the directory to which stellar populations luminosities (integrated under a filter) should be stored to file for rapid reuse.
Defined in: subroutine:Stellar_Population_Luminosity_Tabulate
File: stellar_populations.luminosities.F90

Used by:

Name: `stellarPopulationLuminosityStoreToFile`
Type: boolean 1
Default value: .true.
Description: Specifies whether or not stellar populations luminosities (integrated under a filter) should be stored to file for rapid reuse.
Defined in: `subroutine:Stellar_Population_Luminosity_Tabulate`
File: `stellar_populations.luminosities.F90`
Used by:

Name: `stepsMaximum`
Type: integer 1
Default value: huge(0)
Description: The maximum number of steps to take.
Defined in: `function:particleSwarmConstructorParameters`
File: `posterior_sampling.simulation.particle_swarm.F90`
Used by:

Name: `stepsPerLevel`
Type: integer 1
Default value: 10
Description: The number of steps to take at each tempering level.
Defined in: `function:temperedDifferentialEvolutionConstructorParameters`
File: `posterior_sampling.simulation.tempered_differential_evolution.F90`
Used by:

Name: `stopAfterCount`
Type: real 1
Description: The number of steps to continue after convergence before stopping.
Defined in: `function:stepCountConstructorParameters`
File: `posterior_sampling.stopping_criteria.step_count.F90`
Used by:

Name: `stride`
Type: integer 1
Description: The size of the stride to take over forests.
Defined in: `function:strideConstructorParameters`
File: `tasks.evolve_forests.work_share.stride.F90`
Used by:

Name: `subhaloAngularMomentaMethod`
Type: boolean 1
Default value: var_str('summation')
Description: Specifies how to account for subhalo angular momentum when adding subhalo mass to host halo mass.
Defined in: `function:readConstructorParameters`
File: `merger_trees.construct.read.F90`
Used by:

Name: `subhaloHierarchyDepth`
Type: integer 1
Default value: 2
Description: The depth in the subhalo hierarchy for which to store unevolved subhalo mass functions.
Defined in: `function:conditionalMFConstructorParameters`
File: `merger_trees.operators.conditional_mass_function.F90`
Used by:

Name: `subtractRandomOffset`
Type: boolean 0..1
Default value: .false.
Description: If true, the center-of-mass postions and velocities of the host and satellite are enforced to be matched with the values specified by the offset parameters. If false, due to limited number of particles in the representations, the center-of-mass postions and velocities may deviate slightly from the specified values, i.e. have small random offsets.
Defined in: `function:particulateConstructorParameters`
File: `merger_trees.operators.particulate.F90`
Used by:

Name: `subvolumeBuffer`
Type: boolean 1
Default value: 0.0
Description: Specifies the buffer region (in units of Mpc/h to follow the format convention) around subvolumes of a "Sussing Merger Trees" format [Srisawat et al., 2013] merger tree file which should be read in to ensure that no halos are missed from trees.
Defined in: `function:sussingConstructorParameters`
File: `merger_trees.construct.read.importer.SussingMergerTrees.F90`
Used by:

Name: `subvolumeCount`
Type: boolean 1
Default value: 1
Description: Specifies the number of subvolumes *along each axis* into which a "Sussing Merger Trees" format [Srisawat et al., 2013] merger tree files should be split for processing through Galacticus.
Defined in: `function:sussingConstructorParameters`
File: `merger_trees.construct.read.importer.SussingMergerTrees.F90`
Used by:

Name: `subvolumeIndex`
Type: boolean 1
Default value: [0,0,0]
Description: Specifies the index (in each dimension) of the subvolume of a "Sussing Merger Trees" format [Srisawat et al., 2013] merger tree file to process. Indices range from 0 to `[subvolumeCount]`-1.
Defined in: `function:sussingConstructorParameters`
File: `merger_trees.construct.read.importer.SussingMergerTrees.F90`
Used by:

Name: `supernovaEnergy`

Type: real 1
Default value: 1.0×10^{51}
Description: The energy produced by a supernova (in ergs).
Defined in: function:standardConstructorParameters
File: stellar_astrophysics.feedback.standard.F90
Used by:

Name: surfaceBrightnessLimit
Type: real 1
Description: The limiting surface brightness to which to integrate.
Defined in: function:massFunctionConstructorParameters
File: models.likelihoods.mass_function.F90
Used by:

Name: surfaceDensityCritical
Type: real 1
Default value: 200.0 [Bigiel et al., 2008]
Description: The surface density (in units of $M_\odot$ pc^{-2}) in the "Blitz-Rosolowsky2006" star formation timescale calculation at which low-density truncation begins.
Defined in: function:blitz2006ConstructorParameters
File: star_formation.rate_surface_density.disks.Blitz2006.F90
Used by:

Name: surfaceDensityExponent
Type: real 1
Default value: 0.4 [Bigiel et al., 2008]
Description: The exponent for surface density in the "Blitz-Rosolowsky2006" star formation timescale calculation at in the high density regime.
Defined in: function:blitz2006ConstructorParameters
File: star_formation.rate_surface_density.disks.Blitz2006.F90
Used by:

Name: surveyRedshiftMaximum
Type: boolean 1
Default value: 0.1
Description: The maximum redshift at which calculations of the mass function covariance should be carried out.
Defined in: function:massFunctionCovarianceConstructorParameters
File: tasks.mass_function_covariance.F90
Used by:

Name: surveyRedshiftMinimum
Type: boolean 1
Default value: 0.0
Description: The minimum redshift at which calculations of the mass function covariance should be carried out.
Defined in: function:massFunctionCovarianceConstructorParameters
File: tasks.mass_function_covariance.F90
Used by:

Name: suspendPath
Type: string 1
Description: The path to which tree suspension files will be stored.
Defined in: function:evolveForestsConstructorParameters
File: tasks.evolve_forests.F90
Used by:

Name: suspendToRAM
Type: boolean 1
Default value: .true.
Description: Specifies whether trees should be suspended to RAM (otherwise they are suspend to file).
Defined in: function:evolveForestsConstructorParameters
File: tasks.evolve_forests.F90
Used by:

Name: system
Type: real 1..*
Description: The photometric system (AB or Vega) of the broad-band SED.
Defined in: function:sedFitConstructorParameters
File: models.likelihoods.SED_fit.F90
Used by:

Name: systematicErrorPolynomialCoefficient
Type: float 0..1
Default value: [0.0d0]
Description: The coefficients of the systematic error polynomial for stellar vs halo mass relation.
Defined in: function:stellarVsHaloMassRelationLeauthaud2012ConstructorParameters
File: galacticus.output.analyses.stellar_vs_halo_mass_relation.COSMOS_Leauthaud2012.F90
Used by:

Name: tablePointsPerDecade
Type: integer 0..1
Default value: 10
Description: The number of points per decade of wavenumber at which to tabulate the transfer function.
Defined in: function:acceleratorConstructorParameters
File: structure_formation.transfer_function.accelerator.F90
Used by:

Name: tableStore
Type: boolean 0..1
Default value: .true.
Description: If true, store/restore the tabulated solution to/from file when possible.
Defined in: function:sphericalCollapseMatterLambdaConstructorParameters
File: structure_formation.virial_density_contrast.spherical_collapse_matter_lambda.F90
Used by:

Name: targetLabel

Type: string 0..1
Default value: var_str('')
Description: A label for the target dataset in a plot of this analysis.
Defined in: function:volumeFunction1DConstructorParameters
File: galacticus.output.analyses.volume_function_1d.F90
Used by:

Name: temperature
Type: real 1
Description: The temperature of the black body radiation field.
Defined in: function:blackBodyConstructorParameters
File: radiation.black_body.F90
Used by:

Name: temperatureCMB
Type: real 0..1
Default value: 2.72548 [Fixsen, 2009]
Description: The present day temperature of the cosmic microwave background (CMB) in units of Kelvin.
Defined in: function:simpleConstructorParameters
File: cosmology.parameters.simple.F90
Used by:

Name: temperatureLevelCount
Type: integer 1
Default value: 10
Description: The number temperature levels to use.
Defined in: function:annealedDifferentialEvolutionConstructorParameters
File: posterior_sampling.simulation.annealed_differential_evolution.F90
Used by:

Name: temperatureMaximum
Type: integer 1
Description: The maximum temperature to reach.
Defined in: function:temperedDifferentialEvolutionConstructorParameters
File: posterior_sampling.simulation.tempered_differential_evolution.F90
Used by:

Name: temperatureScale
Type: integer 1
Description: The temperature scale.
Defined in: function:stochasticDifferentialEvolutionConstructorParameters
File: posterior_sampling.simulation.stochastic_differential_evolution.F90
Used by:

Name: temperingLevelCount
Type: integer 1
Default value: 10
Description: The number tempering levels to use.

Defined in: function:temperedDifferentialEvolutionConstructorParameters
File: posterior_sampling.simulation.tempered_differential_evolution.F90
Used by:

Name: testCount
Type: integer 1
Default value: 10
Description: The interval in number of steps at which to check convergence.
Defined in: function:gelmanRubinConstructorParameters
File: posterior_sampling.convergence.Gelman-Rubin.F90
Used by:

Name: threadLock
Type: boolean 1
Default value: .false.
Description: Specifies whether or not to limit the number of threads across all Galacticus processes.
Defined in: function:evolveForestsConstructorParameters
File: tasks.evolve_forests.F90
Used by:

Name: threadLockName
Type: string 1
Default value: var_str('galacticus')
Description: The name to use for the semaphore used to lock threads across all Galacticus processes.
Defined in: function:evolveForestsConstructorParameters
File: tasks.evolve_forests.F90
Used by:

Name: threadsMaximum
Type: string 1
Default value: var_str('processorCount')
Description: The maximum number of active threads across all Galacticus processes.
Defined in: function:evolveForestsConstructorParameters
File: tasks.evolve_forests.F90
Used by:

Name: thresholdCollapse
Type: real 1
Description: The threshold for collapse of density perturbations.
Defined in: function:peakBackgroundConstructorParameters
File: statistics.distributions.peak_background.F90
Used by:

Name: thresholdHatR
Type: real 1
Default value: 1.2
Description: The $\hat{R}$ value at which convergence is declared.
Defined in: function:gelmanRubinConstructorParameters
File: posterior_sampling.convergence.Gelman-Rubin.F90

Used by:

Name: `thresholdMaximum`
Type: float 0..1
Description: Maximum threshold for the min-max property operator.
Defined in: `function:minMaxConstructorParameters`
File: `galacticus.output.analyses.property_operator.minmax.F90`
Used by:

Name: `thresholdMinimum`
Type: float 0..1
Description: Minimum threshold for the min-max property operator.
Defined in: `function:minMaxConstructorParameters`
File: `galacticus.output.analyses.property_operator.minmax.F90`
Used by:

Name: `thresholdStabilityShock`
Type: real 1
Default value: 0.0126 [Birnboim and Dekel, 2003]
Description: The threshold value, $\epsilon_{\rm s,crit}$, for shock stability in the model of Birnboim and Dekel [2003].
Defined in: `function:coldModeConstructorParameters`
File: `accretion.halo.cold_mode.F90`
Used by:

Name: `timeBegin`
Type: real 1
Default value: 0.05d0*ageUniverse
Description: The earliest time at which to tabulate the evolution of main branch progenitor galaxies (in Gyr).
Defined in: `function:recordEvolutionConstructorParameters`
File: `merger_trees.evolve.timesteps.record_evolution.F90`
Used by:

Name: `timeBinsPerDecade`
Type: integer 1
Default value: 10
Description: The number of bins per decade of time at which to count non-primary progenitors.
Defined in: `function:profilerConstructorParameters`
File: `merger_trees.operators.profiler.F90`
Used by:

Name: `timeCountPerDecade`
Type: integer 1
Default value: 10
Description: The number of bins per decade of time to use for calculations of the properties of the universe.
Defined in: `function:intergalacticMediumStateEvolveConstructorParameters`
File: `universe.operators.intergalactic_medium_state_evolve.F90`
Used by:

Name: `timeEnd`
Type: real 1
Default value: ageUniverse
Description: The latest time at which to tabulate the evolution of main branch progenitor galaxies (in Gyr).
Defined in: function:recordEvolutionConstructorParameters
File: merger_trees.evolve.timesteps.record_evolution.F90
Used by:

Name: `timeEvolvesAlongLightcone`
Type: real 0..1
Default value: .true.
Description: If `true`, cosmic time evolves along the lightcone as expected. Otherwise, time is fixed at the present epoch throughout the lightone. This allows construction of lightcones with no evolution.
Defined in: function:squareConstructorParameters
File: geometry.lightcones.square.F90
Used by:

Name: `timeFine`
Type: real 1
Default value: 0.1
Description: The period prior to each output for which the fine time step is used in tabulations of star formation histories [Gyr].
Defined in: function:metallicitySplitConstructorParameters
File: star_formation.histories.metallicity_split.F90
Used by:

Name: `timeFormationSeekDelta`
Type: real 1
Default value: 0.0
Description: The parameter $\Delta \log t$ by which the logarithm of the trial formation time is incremented when stepping through the formation history of a node to find the formation time. If set to zero (or a negative value) the cumulative mass histories of nodes are assumed to be monotonic functions of time, and the formation time is instead found by a root finding algorithm,
Defined in: function:ludlow2016ConstructorParameters
File: dark_matter_profiles.structure.scale.Ludlow2016.F90
Used by:

Name: `timeLimit`
Type: real 1
Default value: 1.0×10^{-2}
Description: The maximum age of stellar populations to retain in the "recent" spectra postprocessing method.
Defined in: function:recentConstructorParameters
File: stellar_populations.spectra.postprocess.recent.F90
Used by:

Name: `timeOffsetMaximumAbsolute`

Type: real 1
Default value: 0.010
Description: The maximum absolute time difference (in Gyr) allowed between merging pairs of galaxies.
Defined in: function:satelliteConstructorParameters
File: merger_trees.evolve.timesteps.satellite.F90
Used by:

Name: timeOffsetMaximumRelative
Type: real 1
Default value: 0.001
Description: The maximum time difference (relative to the cosmic time at the merger epoch) allowed between merging pairs of galaxies.
Defined in: function:satelliteConstructorParameters
File: merger_trees.evolve.timesteps.satellite.F90
Used by:

Name: timeRecent
Type: real 0..1
Description: Halos which experienced a major node merger within a time $\Delta t =$[timeRecent] of the analysis time will be excluded from the analysis.
Defined in: function:spinDistributionBett2007ConstructorParameters
File: galacticus.output.analyses.spin_distribution.Bett2007.F90
Used by:

Name: timeScale
Type: real 0..1
Default value: 1.0
Description: The timescale (in Gyr) for cooling in the simple cooling rate model.
Defined in: function:simpleConstructorParameters
File: cooling.cooling_rate.simple.F90
Used by:

Name: timeSnapTolerance
Type: real 1
Default value: 1.0×10^{-6}
Description: The fractional tolerance within which the tree base time will be snapped to a nearby output time.
Defined in: function:buildConstructorParameters
File: merger_trees.construct.build.F90
Used by:

Name: timeSnapshot
Type: real 0..1
Description: The time at which to snapshot the tree.
Defined in: function:particulateConstructorParameters
File: merger_trees.operators.particulate.F90
Used by:

Name: timeStart

Type: float 1
Description: The time at which two-body relaxation is assumed to have begun.
Defined in: function:twoBodyRelaxationConstructorParameters
File: dark_matter_profiles.heating.two_body_relaxation.F90
Used by:

Name: timeStep
Type: real 1
Default value: 0.1
Description: The time step to use in tabulations of star formation histories [Gyr].
Defined in: function:metallicitySplitConstructorParameters
File: star_formation.histories.metallicity_split.F90
Used by:

Name: timeStepAbsolute
Type: real 1
Default value: 1.0
Description: The maximum allowed absolute change in time (in Gyr) for a single step in the evolution of a node.
Defined in: function:simpleConstructorParameters
File: merger_trees.evolve.timesteps.simple.F90
Used by:

Name: timeStepFine
Type: real 1
Default value: 0.01
Description: The fine time step to use in tabulations of star formation histories [Gyr].
Defined in: function:metallicitySplitConstructorParameters
File: star_formation.histories.metallicity_split.F90
Used by:

Name: timeStepFractional
Type: real 0..1
Default value: 0.01
Description: The fractional time step used when computing barrier crossing rates (i.e. the step used in finite difference calculations).
Defined in: function:farahiConstructorParameters
File: structure_formation.excursion_sets.first_crossing_distribution.Farahi.F90
Used by:

Name: timeStepRelative
Type: real 1
Default value: 0.1
Description: The maximum allowed relative change in time for a single step in the evolution of a node.
Defined in: function:simpleConstructorParameters
File: merger_trees.evolve.timesteps.simple.F90
Used by:

Name: times

Type: real 1..*
Description: A list of (space-separated) times at which Galacticus results should be output. Times need not be in any particular order.
Defined in: function:listConstructorParameters
File: output.times.list.F90
Used by:

Name: timesPerDecade
Type: integer 1
Default value: 10
Description: The number of points per decade of time at which to tabulate excursion set solutions.
Defined in: function:excursionSetsConstructorParameters
File: tasks.excursion_sets.F90
Used by:

Name: timescale
Type: real 1
Default value: 1.0×10^{-2}
Description: The timescale for "escape" of stellar populations in the "unescaped" spectra postprocessing method.
Defined in: function:unescapedConstructorParameters
File: stellar_populations.spectra.postprocess.unescaped.F90
Used by:

Name: timescaleMinimum
Type: real 1
Default value: 0.001
Description: The minimum timescale (in Gyr) for cooling the velocity maximum scaling scaling cooling rate model.
Defined in: function:velocityMaximumScalingConstructorParameters
File: cooling.cooling_rate.velocity_maximum_scaling.F90
Used by:

Name: timescaleMultiplier
Type: real 1
Default value: 0.75
Description: A multiplier for the merging timescale in dynamical friction timescale calculations.
Defined in: function:wetzelWhite2010ConstructorParameters
File: satellites.merging.timescale.Wetzel-White.F90
Used by:

Name: timescaleNormalization
Type: real 1
Default value: 0.13
Description: The mass loss timescale normalization (in Gyr) for the van den Bosch et al. [2005] dark matter halo mass loss rate algorithm.
Defined in: function:vanDenBoschConstructorParameters
File: dark_matter_halos.mass_loss_rates.vanDenBosch.F90
Used by:

Name: `timestepHostAbsolute`
Type: real 1
Default value: 1.0
Description: The maximum allowed absolute timestep (in Gyr) for node evolution relative to the time of the host halo.
Defined in: function:standardConstructorParameters
File: merger_trees.evolver.standard.F90
Used by:

Name: `timestepHostRelative`
Type: real 1
Default value: 0.1
Description: The maximum allowed relative timestep for node evolution relative to the time of the host halo.
Defined in: function:standardConstructorParameters
File: merger_trees.evolver.standard.F90
Used by:

Name: `tolerance`
Type: real 1
Default value: 4.0×10^{-6}
Description: The relative tolerance to use in integrating over the linear power spectrum to compute the cosmological mass variance.
Defined in: function:filteredPowerConstructorParameters
File: structure_formation.cosmological_mass_variance.filtered_power_spectrum.F90
Used by:

Name: `toleranceResolutionParent`
Type: real 0..1
Default value: 1.0×10^{-3}
Description: The fractional tolerance in parent node mass at the resolution limit below which branch mis-orderings will be ignored.
Defined in: function:cole2000ConstructorParameters
File: merger_trees.construct.builder.Cole2000.F90
Used by:

Name: `toleranceResolutionSelf`
Type: real 0..1
Default value: 1.0×10^{-6}
Description: The fractional tolerance in node mass at the resolution limit below which branch mis-orderings will be ignored.
Defined in: function:cole2000ConstructorParameters
File: merger_trees.construct.builder.Cole2000.F90
Used by:

Name: `toleranceScale`
Type: real 0..
Default value: 0.15

Description: The tolerance scale used in deciding if a trial tree is an acceptable match.
Defined in: function:augmentConstructorParameters
File: merger_trees.operators.augment.F90
Used by:

Name: toleranceTopHat
Type: real 1
Default value: 1.0×10^{-6}
Description: The relative tolerance to use in integrating over the linear power spectrum using a top-hat (real space) window function to compute the cosmological mass variance.
Defined in: function:filteredPowerConstructorParameters
File: structure_formation.cosmological_mass_variance.filtered_power_spectrum.F90
Used by:

Name: toomreParameterCritical
Type: real 1
Default value: 0.4 [Kennicutt, 1989]
Description: The critical Toomre parameter for star formation in disks.
Defined in: function:kennicuttSchmidtConstructorParameters
File: star_formation.rate_surface_density.disks.Kennicutt-Schmidt.F90
Used by:

Name: topLevel
Type: boolean 1
Default value: .false.
Description: If true, output the index of the host at the top level of the hierarchy, otherwise output the index of the direct host.
Defined in: function:indicesHostConstructorParameters
File: nodes.property_extractor.hosts.F90
Used by:

Name: treeBeginAt
Type: integer 1
Default value: 0
Description: The index (in order of increasing base halo mass) of the tree at which to begin when building merger trees. A value of "0" means to begin with tree number 1 (if processing trees in ascending order), or equal to the number of trees (otherwise).
Defined in: function:buildConstructorParameters
File: merger_trees.construct.build.F90
Used by:

Name: treeCount
Type: float 1
Default value: spread(1,1,fixedHalosCount)
Description: Specifies the number of halos to use when building halos.
Defined in: function:fixedMassConstructorParameters
File: merger_trees.construct.build.masses.fixed_mass.F90
Used by:

Name: `treeIndexToRootNodeIndex`
Type: boolean 1
Default value: .false.
Description: Specifies whether tree indices should always be set to the index of their root node.
Defined in: `function:readConstructorParameters`
File: `merger_trees.construct.read.F90`
Used by:

Name: `treeNodeMethodAgeStatistics`
Type: string 1
Default value: var_str('null')
Description: Specifies the implementation to be used for the ageStatistics component of nodes.
Defined in: `subroutine:nodeClassHierarchyInitialize`
File: `objects.nodes.components.Inc`
Used by:

Name: `treeNodeMethodBasic`
Type: string 1
Default value: var_str('standard')
Description: Specifies the implementation to be used for the basic component of nodes.
Defined in: `subroutine:nodeClassHierarchyInitialize`
File: `objects.nodes.components.Inc`
Used by:

Name: `treeNodeMethodBlackHole`
Type: string 1
Default value: var_str('standard')
Description: Specifies the implementation to be used for the blackHole component of nodes.
Defined in: `subroutine:nodeClassHierarchyInitialize`
File: `objects.nodes.components.Inc`
Used by:

Name: `treeNodeMethodDarkMatterProfile`
Type: string 1
Default value: var_str('scale')
Description: Specifies the implementation to be used for the darkMatterProfile component of nodes.
Defined in: `subroutine:nodeClassHierarchyInitialize`
File: `objects.nodes.components.Inc`
Used by:

Name: `treeNodeMethodDisk`
Type: string 1
Default value: var_str('standard')
Description: Specifies the implementation to be used for the disk component of nodes.
Defined in: `subroutine:nodeClassHierarchyInitialize`
File: `objects.nodes.components.Inc`
Used by:

Name: `treeNodeMethodDynamicsStatistics`

Type: string 1
Default value: var_str('null')
Description: Specifies the implementation to be used for the dynamicsStatistics component of nodes.
Defined in: `subroutine:nodeClassHierarchyInitialize`
File: `objects.nodes.components.Inc`
Used by:

Name: `treeNodeMethodFormationTime`
Type: string 1
Default value: var_str('null')
Description: Specifies the implementation to be used for the formationTime component of nodes.
Defined in: `subroutine:nodeClassHierarchyInitialize`
File: `objects.nodes.components.Inc`
Used by:

Name: `treeNodeMethodHostHistory`
Type: string 1
Default value: var_str('null')
Description: Specifies the implementation to be used for the hostHistory component of nodes.
Defined in: `subroutine:nodeClassHierarchyInitialize`
File: `objects.nodes.components.Inc`
Used by:

Name: `treeNodeMethodHotHalo`
Type: string 1
Default value: var_str('standard')
Description: Specifies the implementation to be used for the hotHalo component of nodes.
Defined in: `subroutine:nodeClassHierarchyInitialize`
File: `objects.nodes.components.Inc`
Used by:

Name: `treeNodeMethodIndices`
Type: string 1
Default value: var_str('null')
Description: Specifies the implementation to be used for the indices component of nodes.
Defined in: `subroutine:nodeClassHierarchyInitialize`
File: `objects.nodes.components.Inc`
Used by:

Name: `treeNodeMethodInterOutput`
Type: string 1
Default value: var_str('null')
Description: Specifies the implementation to be used for the interOutput component of nodes.
Defined in: `subroutine:nodeClassHierarchyInitialize`
File: `objects.nodes.components.Inc`
Used by:

Name: `treeNodeMethodMassFlowStatistics`
Type: string 1

Default value: var_str('null')
Description: Specifies the implementation to be used for the massFlowStatistics component of nodes.
Defined in: `subroutine:nodeClassHierarchyInitialize`
File: `objects.nodes.components.Inc`
Used by:

Name: `treeNodeMethodMergingStatistics`
Type: string 1
Default value: var_str('null')
Description: Specifies the implementation to be used for the mergingStatistics component of nodes.
Defined in: `subroutine:nodeClassHierarchyInitialize`
File: `objects.nodes.components.Inc`
Used by:

Name: `treeNodeMethodNBody`
Type: string 1
Default value: var_str('null')
Description: Specifies the implementation to be used for the nBody component of nodes.
Defined in: `subroutine:nodeClassHierarchyInitialize`
File: `objects.nodes.components.Inc`
Used by:

Name: `treeNodeMethodPosition`
Type: string 1
Default value: var_str('null')
Description: Specifies the implementation to be used for the position component of nodes.
Defined in: `subroutine:nodeClassHierarchyInitialize`
File: `objects.nodes.components.Inc`
Used by:

Name: `treeNodeMethodSatellite`
Type: string 1
Default value: var_str('standard')
Description: Specifies the implementation to be used for the satellite component of nodes.
Defined in: `subroutine:nodeClassHierarchyInitialize`
File: `objects.nodes.components.Inc`
Used by:

Name: `treeNodeMethodSpheroid`
Type: string 1
Default value: var_str('standard')
Description: Specifies the implementation to be used for the spheroid component of nodes.
Defined in: `subroutine:nodeClassHierarchyInitialize`
File: `objects.nodes.components.Inc`
Used by:

Name: `treeNodeMethodSpin`
Type: string 1
Default value: var_str('random')

Description: Specifies the implementation to be used for the spin component of nodes.
Defined in: subroutine:nodeClassHierarchyInitialize
File: objects.nodes.components.Inc
Used by:

Name: treeSampleRate
Type: float 1
Default value: 1.0
Description: Specify the probability that any given tree should processed (to permit subsampling).
Defined in: function:sussingConstructorParameters
File: merger_trees.construct.read.importer.SussingMergerTrees.F90
Used by:

Name: treesPerDecade
Type: real 1
Default value: 10.0
Description: The number of merger trees masses to sample per decade of base halo mass.
Defined in: function:sampledDistributionConstructorParameters
File: merger_trees.construct.build.masses.sampled_distribution.F90
Used by:

Name: tripleBlackHoleInteraction
Type: boolean 1
Default value: .false.
Description: Determines whether or not triple black hole interactions will be accounted for.
Defined in: subroutine:Node_Component_Black_Hole_Noncentral_Initialize
File: objects.nodes.components.black_hole.non_central.F90
Used by:

Name: truncate
Type: boolean 1
Default value: .true.
Description: Specifies whether or not to truncate star formation below a critical surface density in disks.
Defined in: function:kennicuttSchmidtConstructorParameters
File: star_formation.rate_surface_density.disks.Kennicutt-Schmidt.F90
Used by:

Name: type
Type: string 1
Description: The metadata type.
Defined in: function:mergerTreeFileBuilderConstructorParameters
File: tasks.merger_tree_file_builder.F90
Used by:

Name: unitLengthInSI
Type: float 0..1
Description: The length unit expressed in the SI system.
Defined in: function:gadgetHDF5ConstructorParameters

File: nBody.import.GadgetHDF5.F90
Used by:

Name: unitMassInSI
Type: float 0..1
Description: The mass unit expressed in the SI system.
Defined in: function:gadgetHDF5ConstructorParameters
File: nBody.import.GadgetHDF5.F90
Used by:

Name: unitVector1
Type: real 0..1
Description: The first (radial) unit vector defining the lightcone geometry.
Defined in: function:squareConstructorParameters
File: geometry.lightcones.square.F90
Used by:

Name: unitVector2
Type: real 0..1
Description: The second (angular) unit vector defining the lightcone geometry.
Defined in: function:squareConstructorParameters
File: geometry.lightcones.square.F90
Used by:

Name: unitVector3
Type: real 0..1
Description: The third (angular) unit vector defining the lightcone geometry.
Defined in: function:squareConstructorParameters
File: geometry.lightcones.square.F90
Used by:

Name: unitVelocityInSI
Type: float 0..1
Description: The velocity unit expressed in the SI system.
Defined in: function:gadgetHDF5ConstructorParameters
File: nBody.import.GadgetHDF5.F90
Used by:

Name: unitsInSI
Type: float 1
Description: The velocity unit in the SI system.
Defined in: function:mergerTreeFileBuilderConstructorParameters
File: tasks.merger_tree_file_builder.F90
Used by:

Name: untemperedStepCount
Type: integer 1
Default value: 10
Description: The number of untempered steps to take.

Defined in: function:temperedDifferentialEvolutionConstructorParameters
File: posterior_sampling.simulation.tempered_differential_evolution.F90
Used by:

Name: updateCount
Type: real 1
Description: The number of steps between potential updates of the temperature exponent.
Defined in: function:adaptiveConstructorParameters
File: posterior_sampling.differential_proposal_size.temperature_exponent.adaptive.F90
Used by:

Name: useFittingFunction
Type: boolean 1
Default value: .true.
Description: Specifies whether the warm dark matter critical overdensity mass scaling should be computed from a fitting function or from tabulated data.
Defined in: function:barkana2001WDMConstructorParameters
File: structure_formation.critical_overdensity.warm_dark_matter.F90
Used by:

Name: useFormationHalo
Type: boolean 1
Default value: .false.
Description: Specifies whether or not the "formation halo" should be used when solving for the radii of galaxies.
Defined in: function:simpleConstructorParameters
File: galactic_structure.radius_solver.simple.F90
Used by:

Name: useFormationNode
Type: real 1
Default value: .false.
Description: Specifies whether to use the virial velocity of the formation node or current node in the cooling rate "cut-off" modifier.
Defined in: function:cutOffConstructorParameters
File: cooling.cooling_rate.cut_off.F90
Used by:

Name: useInteriorMean
Type: boolean 1
Default value: .false.
Description: Specifies whether to use the specific angular momentum at the cooling radius, or the mean specific angular momentum interior to that radius.
Defined in: function:constantRotationConstructorParameters
File: cooling.specific_angular_momentum.constant_rotation.F90
Used by:

Name: useMeanConcentration
Type: string 1

Default value: .false.
Description: If true, then when computing dark matter profile scale radii using concentrations do not account for any possible scatter in the concentration-mass relation.
Defined in: function:concentrationConstructorParameters
File: dark_matter_profiles.structure.scale.concentration.F90
Used by:

Name: useOneNodeTrees
Type: boolean 0..1
Default value: .false.
Description: If true, trees only consisting of their base node will be augmented.
Defined in: function:augmentConstructorParameters
File: merger_trees.operators.augment.F90
Used by:

Name: useSurveyLimits
Type: boolean 1
Default value: .false.
Description: Specifies whether the limiting redshifts for integrating over the halo mass function should be limited by those of a galaxy survey.
Defined in: function:conditionalMassFunctionConstructorParameters
File: tasks.conditional_mass_function.F90
Used by:

Name: useVelocityMostBound
Type: boolean 0..1
Default value: .false.
Description: If true, the velocity of the most bound particle in velocity space is used as the representative velocity of the satellite. If false, use the mass weighted mean velocity (center-of-mass velocity) of self-bounf particles instead.
Defined in: function:selfBoundConstructorParameters
File: nBody.operator.self_bound.F90
Used by:

Name: variance
Type: real 1
Description: The variance of the normal distribution.
Defined in: function:normalConstructorParameters
File: statistics.distributions.normal.F90
Used by:

Name: varianceBackground
Type: real 1
Description: The variance in the background density field.
Defined in: function:peakBackgroundConstructorParameters
File: statistics.distributions.peak_background.F90
Used by:

Name: vectorRotation

Type: float 1
Default value: [acos(2.0d0*randomSequence%uniformSample()-1.0d0),2.0d0*Pi*randomSequence%uniformSample()] Isotropically random on the unit sphere.
Description: The vector, in spherical coordinates (θ, ϕ), about which the mock catalog should be rotated.
Defined in: function:catalogProjectedCorrelationFunctionConstructorParameters
File: tasks.catalog_projected_correlation_function.F90
Used by:

Name: velocityCharacteristic
Type: real 0..1
Default value: 250.0
Description: The velocity scale at which the supernovae (SNe)-driven outflow rate equals the star formation rate in disks.
Defined in: function:powerLawConstructorParameters
File: star_formation.feedback.disks.power_law.F90
Used by:

Name: velocityCoefficient
Type: real 1
Default value: 0.5
Description: Velocity parameter.
Defined in: function:particleSwarmConstructorParameters
File: posterior_sampling.simulation.particle_swarm.F90
Used by:

Name: velocityCutOff
Type: real 1
Default value: 200.0
Description: The halo maximum velocity scale appearing in the exponential term for cooling timescale in the velocity maximum scaling cooling rate model.
Defined in: function:velocityMaximumScalingConstructorParameters
File: cooling.cooling_rate.velocity_maximum_scaling.F90
Used by:

Name: velocityCutOffExponentRedshift
Type: real 1
Default value: 0.0
Description: The exponent of $(1+z)$ in the velocity scale appearing in the exponential term for cooling timescale in the velocity maximum scaling cooling rate model.
Defined in: function:velocityMaximumScalingConstructorParameters
File: cooling.cooling_rate.velocity_maximum_scaling.F90
Used by:

Name: velocityDispersionDiskGas
Type: real 1
Default value: 10.0 [Leroy et al., 2008]
Description: The velocity dispersion of gas in disks.
Defined in: function:kennicuttSchmidtConstructorParameters

File: star_formation.rate_surface_density.disks.Kennicutt-Schmidt.F90
Used by:

Name: velocityExponent
Type: real 1
Default value: 0.0
Description: The exponent of maximum circular velocity in the velocity maximum scaling model for outflow reincorporation.
Defined in: function:velocityMaximumScalingConstructorParameters
File: hot_halo.outflow_reincorporation.velocity_maximum_scaling.F90
Used by:

Name: velocityRadial
Type: real 1
Default value: -0.90d0
Description: The radial velocity (in units of the host virial velocity) to used for the fixed virial orbits distribution. Default value matches approximate peak in the distribution of Benson [2005].
Defined in: function:fixedConstructorParameters
File: satellites.merging.virial_orbits.fixed.F90
Used by:

Name: velocitySuppressionReionization
Type: real 1
Default value: 35.0
Description: The velocity scale below which baryonic accretion is suppressed.
Defined in: function:simpleConstructorParameters
File: accretion.halo.simple.F90
Used by:

Name: velocityTangential
Type: real 1
Default value: 0.75
Description: The tangential velocity (in units of the host virial velocity) to used for the fixed virial orbits distribution. Default value matches approximate peak in the distribution of Benson [2005].
Defined in: function:fixedConstructorParameters
File: satellites.merging.virial_orbits.fixed.F90
Used by:

Name: verbosityLevel
Type: integer 1
Default value: 1
Description: The level of verbosity for Galacticus (higher values give more verbosity).
Defined in: subroutine:Galacticus_Verbosity_Set_From_Parameters
File: galacticus.display.verbosity.F90
Used by:

Name: virialRadius
Type: real 1
Default value: 1.0

Description: The virial radius of the NFW profile.
Defined in: function:nfwConstructorParameters
File: mass_distributions.spherical.NFW.F90
Used by:

Name: viscosityAlpha
Type: real 1
Default value: 0.1
Description: The value for the viscosity parameter α in an ADAF to be used if [adafViscosityOption]=fixed.
Defined in: function:adafConstructorParameters
File: accretion_disks.ADAF.F90
Used by:

Name: viscosityOption
Type: string 1
Default value: var_str('fit')
Description: Controls how the viscosity parameter α in an ADAF is determined. fit will cause α to be computed using the fitting function of Benson and Babul [2009]; fixed will cause α =[adafViscosityFixedAlpha] to be used.
Defined in: function:adafConstructorParameters
File: accretion_disks.ADAF.F90
Used by:

Name: walltimeMaximum
Type: boolean 1
Default value: -1_kind_int8
Description: If set to a positive number, this is the maximum wall time for which forest evolution is allowed to proceed before the task gives up.
Defined in: function:evolveForestsConstructorParameters
File: tasks.evolve_forests.F90
Used by:

Name: wavelengthCountPerDecade
Type: integer 1
Default value: 10
Description: The number of bins per decade of wavelength to use for calculations of the cosmic background radiation.
Defined in: function:intergalacticBackgroundInternalConstructorParameters
File: radiation.intergalactic_background.internal.F90
Used by:

Name: wavelengthExponent
Type: string 0..1
Default value: 0.7
Description: Exponent of wavelength in the optical depth.
Defined in: function:lmnstyStllrChrltFll2000ConstructorParameters
File: nodes.property_extractor.luminosity_stellar.dust.CharlotFall2000.F90
Used by:

Name: `wavelengthMaximum`
Type: real 1
Default value: 100000.0
Description: The maximum wavelength (in units of Å) to use in calculations of the cosmic background radiation.
Defined in: function:intergalacticBackgroundInternalConstructorParameters
File: radiation.intergalactic_background.internal.F90
Used by:

Name: `wavelengthMinimum`
Type: real 1
Default value: 100.0
Description: The minimum wavelength (in units of Å) to use in calculations of the cosmic background radiation.
Defined in: function:intergalacticBackgroundInternalConstructorParameters
File: radiation.intergalactic_background.internal.F90
Used by:

Name: `wavenumberCount`
Type: integer 0..1
Default value: 60_c_size_t
Description: The number of bins in wavenumber to use in computing the correlation function.
Defined in: function:correlationFunctionConstructorParameters
File: galacticus.output.analyses.correlation_function.F90
Used by:

Name: `wavenumberMaximum`
Type: real 1
Default value: 1.0×10^{3}
Description: The maximum wavenumber at which to tabulate power spectra.
Defined in: function:powerSpectraConstructorParameters
File: tasks.power_spectrum.F90
Used by:

Name: `wavenumberMinimum`
Type: real 1
Default value: 1.0×10^{-3}
Description: The minimum wavenumber at which to tabulate power spectra.
Defined in: function:powerSpectraConstructorParameters
File: tasks.power_spectrum.F90
Used by:

Name: `wavenumberReference`
Type: real 0..1
Default value: 1.0
Description: When a running power spectrum index is used, this is the wavenumber at which the index is equal to `[index]`.
Defined in: function:powerLawConstructorParameters
File: structure_formation.power_spectrum.primordial.power_law.F90

Used by:

Name: `whenCutOff`
Type: real 1
Default value: var_str('after')
Description: Specifies whether cooling is cut off before or after `[redshiftCutOff]`.
Defined in: function:cutOffConstructorParameters
File: cooling.cooling_rate.cut_off.F90
Used by:

Name: `widthBuffer`
Type: float 1
Default value: 30.0
Description: The width of the buffer region around survey geometry to ensure galaxies are not lost when moving to redshift space.
Defined in: function:catalogProjectedCorrelationFunctionConstructorParameters
File: tasks.catalog_projected_correlation_function.F90
Used by:

Name: `widthCutOff`
Type: real 1
Default value: 1.0
Description: The width appearing in the exponential term for cooling timescale in the velocity maximum scaling scaling cooling rate model.
Defined in: function:velocityMaximumScalingConstructorParameters
File: cooling.cooling_rate.velocity_maximum_scaling.F90
Used by:

Name: `widthTransitionStabilityShock`
Type: real 1
Default value: 0.01 [Benson and Bower, 2010]
Description: The width of the transition from stability to instability for cold mode accretion [Benson and Bower, 2010].
Defined in: function:coldModeConstructorParameters
File: accretion.halo.cold_mode.F90
Used by:

Name: `xAxisIsLog`
Type: boolean 0..1
Description: If true, indicates that the x-axis should be logarithmic in a plot of this analysis.
Defined in: function:volumeFunction1DConstructorParameters
File: galacticus.output.analyses.volume_function_1d.F90
Used by:

Name: `xAxisLabel`
Type: string 0..1
Description: A label for the x-axis in a plot of this analysis.
Defined in: function:volumeFunction1DConstructorParameters
File: galacticus.output.analyses.volume_function_1d.F90

Used by:

Name: yAxisIsLog
Type: boolean 0..1
Description: If true, indicates that the y-axis should be logarithmic in a plot of this analysis.
Defined in: function:volumeFunction1DConstructorParameters
File: galacticus.output.analyses.volume_function_1d.F90
Used by:

Name: yAxisLabel
Type: string 0..1
Description: A label for the y-axis in a plot of this analysis.
Defined in: function:volumeFunction1DConstructorParameters
File: galacticus.output.analyses.volume_function_1d.F90
Used by:

Name: zeroPoint
Type: string 1
Description: Mass zero point for the mass function incompleteness surface brightness model, i.e. M_0 in $\mu(M) = \alpha \log_{10}(M/M_0) + \beta$.
Defined in: function:surfaceBrightnessConstructorParameters
File: statistics.mass_function.incompleteness.surface_brightness.F90
Used by:

Name: zeta
Type: real 1
Default value: 0.36
Description: The mass loss scaling with halo mass for the van den Bosch et al. [2005] dark matter halo mass loss rate algorithm.
Defined in: function:vanDenBoschConstructorParameters
File: dark_matter_halos.mass_loss_rates.vanDenBosch.F90
Used by:

Name: accretionDiskSpectraMethod
Attached to: module:Accretion_Disk_Spectra
File: accretion_disks.spectra.F90
Default value: hopkins2007
Description: The method to be used for accretionDiskSpectra.

Name: accretionDisksMethod
Attached to: module:Accretion_Disks
File: accretion_disks.F90
Default value: shakuraSunyaev
Description: The method to be used for accretionDisks.

Name: accretionHaloMethod
Attached to: module:Accretion_Halos
File: accretion.halo.F90
Default value: simple

Description: The method to be used for `accretionHalo`.

Name: `accretionHaloTotalMethod`
Attached to: `module:Accretion_Halo_Totals`
File: `accretion.halo.total.F90`
Default value: simple
Description: The method to be used for `accretionHaloTotal`.

Name: `atomicCrossSectionIonizationPhotoMethod`
Attached to: `module:Atomic_Cross_Sections_Ionization_Photo`
File: `atomic.cross_sections.ionization.photo.F90`
Default value: verner
Description: The method to be used for `atomicCrossSectionIonizationPhoto`.

Name: `atomicExcitationRateCollisionalMethod`
Attached to: `module:Atomic_Rates_Excitation_Collisional`
File: `atomic.rates.excitation.collisional.F90`
Default value: scholzWalters1991
Description: The method to be used for `atomicExcitationRateCollisional`.

Name: `atomicIonizationPotentialMethod`
Attached to: `module:Atomic_Ionization_Potentials`
File: `atomic.ionization_potentials.F90`
Default value: verner
Description: The method to be used for `atomicIonizationPotential`.

Name: `atomicIonizationRateCollisionalMethod`
Attached to: `module:Atomic_Rates_Ionization_Collisional`
File: `atomic.rates.ionization.collisional.F90`
Default value: verner1996
Description: The method to be used for `atomicIonizationRateCollisional`.

Name: `atomicRecombinationRateDielectronicMethod`
Attached to: `module:Atomic_Rates_Recombination_Dielectronic`
File: `atomic.rates.recombination.dielectronic.F90`
Default value: arnaud1985
Description: The method to be used for `atomicRecombinationRateDielectronic`.

Name: `atomicRecombinationRateRadiativeMethod`
Attached to: `module:Atomic_Rates_Recombination_Radiative`
File: `atomic.rates.recombination.radiative.F90`
Default value: verner1996
Description: The method to be used for `atomicRecombinationRateRadiative`.

Name: `blackHoleBinaryInitialSeparationMethod`
Attached to: `module:Black_Hole_Binary_Initial_Separation`
File: `black_holes.binaries.initial_separation.F90`
Default value: spheroidRadiusFraction
Description: The method to be used for `blackHoleBinaryInitialSeparation`.

Name: blackHoleBinaryMergerMethod
Attached to: module:Black_Hole_Binary_Mergers
File: black_holes.binary_mergers.F90
Default value: rezzolla2008
Description: The method to be used for blackHoleBinaryMerger.

Name: blackHoleBinaryRecoilMethod
Attached to: module:Black_Hole_Binary_Recoil_Velocities
File: black_holes.binaries.recoil_velocity.F90
Default value: zero
Description: The method to be used for blackHoleBinaryRecoil.

Name: blackHoleBinarySeparationGrowthRateMethod
Attached to: module:Black_Hole_Binary_Separations
File: black_holes.binaries.separation_growth_rate.F90
Default value: zero
Description: The method to be used for blackHoleBinarySeparationGrowthRate.

Name: chemicalReactionRateMethod
Attached to: module:Chemical_Reaction_Rates
File: chemical.reaction_rates.F90
Default value: zero
Description: The method to be used for chemicalReactionRate.

Name: chemicalStateMethod
Attached to: module:Chemical_States
File: chemical.state.F90
Default value: atomicCIECloudy
Description: The method to be used for chemicalState.

Name: coldModeInfallRateMethod
Attached to: module:Cooling_Cold_Mode_Infall_Rates
File: cooling.cold_mode.infall_rate.F90
Default value: dynamicalTime
Description: The method to be used for coldModeInfallRate.

Name: conditionalMassFunctionMethod
Attached to: module:Conditional_Mass_Functions
File: halo_model.conditional_mass_function.F90
Default value: behroozi2010
Description: The method to be used for conditionalMassFunction.

Name: coolingFunctionMethod
Attached to: module:Cooling_Functions
File: cooling.cooling_function.F90
Default value: atomicCIECloudy
Description: The method to be used for coolingFunction.

Name: coolingInfallRadiusMethod
Attached to: module:Cooling_Infall_Radii
File: cooling.infall_radius.F90
Default value: coolingRadius
Description: The method to be used for coolingInfallRadius.

Name: coolingRadiusMethod
Attached to: module:Cooling_Radii
File: cooling.cooling_radius.F90
Default value: simple
Description: The method to be used for coolingRadius.

Name: coolingRateMethod
Attached to: module:Cooling_Rates
File: cooling.cooling_rate.F90
Default value: whiteFrenk1991
Description: The method to be used for coolingRate.

Name: coolingSpecificAngularMomentumMethod
Attached to: module:Cooling_Specific_Angular_Momenta
File: cooling.specific_angular_momentum.F90
Default value: constantRotation
Description: The method to be used for coolingSpecificAngularMomentum.

Name: coolingTimeAvailableMethod
Attached to: module:Cooling_Times_Available
File: cooling.time_available.F90
Default value: whiteFrenk1991
Description: The method to be used for coolingTimeAvailable.

Name: coolingTimeMethod
Attached to: module:Cooling_Times
File: cooling.cooling_time.F90
Default value: simple
Description: The method to be used for coolingTime.

Name: cosmologicalMassVarianceMethod
Attached to: module:Cosmological_Density_Field
File: structure_formation.cosmological_density_field.F90
Default value: filteredPower
Description: The method to be used for cosmologicalMassVariance.

Name: cosmologyFunctionsMethod
Attached to: module:Cosmology_Functions
File: cosmology.functions.F90
Default value: matterLambda
Description: The method to be used for cosmologyFunctions.

Name: cosmologyParametersMethod

Attached to: module:Cosmology_Parameters
File: cosmology.parameters.F90
Default value: simple
Description: The method to be used for `cosmologyParameters`.

Name: `criticalOverdensityMethod`
Attached to: module:Cosmological_Density_Field
File: structure_formation.cosmological_density_field.F90
Default value: sphericalCollapseMatterLambda
Description: The method to be used for `criticalOverdensity`.

Name: `darkMatterHaloBiasMethod`
Attached to: module:Dark_Matter_Halo_Biases
File: structure_formation.halo_bias.F90
Default value: tinker2010
Description: The method to be used for `darkMatterHaloBias`.

Name: `darkMatterHaloMassAccretionHistoryMethod`
Attached to: module:Dark_Matter_Halo_Mass_Accretion_Histories
File: dark_matter_halos.mass_accretion_history.F90
Default value: wechsler2002
Description: The method to be used for `darkMatterHaloMassAccretionHistory`.

Name: `darkMatterHaloMassLossRateMethod`
Attached to: module:Dark_Matter_Halos_Mass_Loss_Rates
File: dark_matter_halos.mass_loss_rates.F90
Default value: zero
Description: The method to be used for `darkMatterHaloMassLossRate`.

Name: `darkMatterHaloScaleMethod`
Attached to: module:Dark_Matter_Halo_Scales
File: dark_matter_halos.scales.F90
Default value: virialDensityContrastDefinition
Description: The method to be used for `darkMatterHaloScale`.

Name: `darkMatterParticleMethod`
Attached to: module:Dark_Matter_Particles
File: dark_matter.particle.F90
Default value: CDM
Description: The method to be used for `darkMatterParticle`.

Name: `darkMatterProfileConcentrationMethod`
Attached to: module:Dark_Matter_Profiles_Concentration
File: dark_matter_profiles.structure.concentration.F90
Default value: gao2008
Description: The method to be used for `darkMatterProfileConcentration`.

Name: `darkMatterProfileDMOMethod`
Attached to: module:Dark_Matter_Profiles_DMO

File: dark_matter_profiles_DMO.F90
Default value: NFW
Description: The method to be used for darkMatterProfileDMO.

Name: darkMatterProfileHeatingMethod
Attached to: module:Dark_Matter_Profiles_DMO
File: dark_matter_profiles_DMO.F90
Default value: null
Description: The method to be used for darkMatterProfileHeating.

Name: darkMatterProfileMethod
Attached to: module:Dark_Matter_Profiles
File: dark_matter_profiles.F90
Default value: adiabaticGnedin2004
Description: The method to be used for darkMatterProfile.

Name: darkMatterProfileScaleRadiusMethod
Attached to: module:Dark_Matter_Profile_Scales
File: dark_matter_profiles.structure.scale.F90
Default value: concentration
Description: The method to be used for darkMatterProfileScaleRadius.

Name: darkMatterProfileShapeMethod
Attached to: module:Dark_Matter_Profiles_Shape
File: dark_matter_profiles.structure.shape.F90
Default value: gao2008
Description: The method to be used for darkMatterProfileShape.

Name: distributionFunction1DMethod
Attached to: module:Statistics_Distributions
File: statistics.distributions.F90
Default value: uniform
Description: The method to be used for distributionFunction1D.

Name: distributionFunctionDiscrete1DMethod
Attached to: module:Statistics_Distributions_Discrete
File: statistics.distributions.discrete.F90
Default value: binomial
Description: The method to be used for distributionFunctionDiscrete1D.

Name: evolveForestsWorkShareMethod
Attached to: module:Task_Evolve_Forests_Work_Shares
File: tasks.evolve_forests.work_share.F90
Default value: FCFS
Description: The method to be used for evolveForestsWorkShare.

Name: excursionSetBarrierMethod
Attached to: module:Excursion_Sets_Barriers
File: structure_formation.excursion_sets.barrier.F90

Default value: criticalOverdensity
Description: The method to be used for `excursionSetBarrier`.

Name: `excursionSetFirstCrossingMethod`
Attached to: `module:Excursion_Sets_First_Crossings`
File: `structure_formation.excursion_sets.first_crossing_distribution.F90`
Default value: linearBarrier
Description: The method to be used for `excursionSetFirstCrossing`.

Name: `freefallRadiusMethod`
Attached to: `module:Freefall_Radii`
File: `cooling.freefall_radii.F90`
Default value: darkMatterHalo
Description: The method to be used for `freefallRadius`.

Name: `freefallTimeAvailableMethod`
Attached to: `module:Cooling_Freefall_Times_Available`
File: `cooling.freefall_time_available.F90`
Default value: haloFormation
Description: The method to be used for `freefallTimeAvailable`.

Name: `galacticDynamicsBarInstabilityMethod`
Attached to: `module:Galactic_Dynamics_Bar_Instabilities`
File: `galactic_dynamics.bar_instability.F90`
Default value: efstathiou1982
Description: The method to be used for `galacticDynamicsBarInstability`.

Name: `galacticFilterMethod`
Attached to: `module:Galactic_Filters`
File: `galactic.filters.F90`
Default value: always
Description: The method to be used for `galacticFilter`.

Name: `galacticStructureSolverMethod`
Attached to: `module:Galactic_Structure_Solvers`
File: `galactic_structure.radius_solver.F90`
Default value: equilibrium
Description: The method to be used for `galacticStructureSolver`.

Name: `gauntFactorMethod`
Attached to: `module:Atomic_Radiation_Gaunt_Factors`
File: `atomic.radiation.gaunt_factors.F90`
Default value: sutherland1998
Description: The method to be used for `gauntFactor`.

Name: `geometryLightconeMethod`
Attached to: `module:Geometry_Lightcones`
File: `geometry.lightcones.F90`
Default value: square

Description: The method to be used for `geometryLightcone`.

Name: `gravitationalLensingMethod`
Attached to: `module:Gravitational_Lensing`
File: `structure_formation.gravitational_lensing.F90`
Default value: takahashi2011
Description: The method to be used for `gravitationalLensing`.

Name: `haloEnvironmentMethod`
Attached to: `module:Cosmological_Density_Field`
File: `structure_formation.cosmological_density_field.F90`
Default value: uniform
Description: The method to be used for `haloEnvironment`.

Name: `haloMassFunctionMethod`
Attached to: `module:Halo_Mass_Functions`
File: `structure_formation.halo_mass_function.F90`
Default value: tinker2008
Description: The method to be used for `haloMassFunction`.

Name: `haloModelPowerSpectrumModifierMethod`
Attached to: `module:Halo_Model_Power_Spectrum_Modifiers`
File: `halo_model.power_spectrum_modifier.F90`
Default value: identity
Description: The method to be used for `haloModelPowerSpectrumModifier`.

Name: `haloSpinDistributionMethod`
Attached to: `module:Halo_Spin_Distributions`
File: `dark_matter_halos.spins.distributions.F90`
Default value: bett2007
Description: The method to be used for `haloSpinDistribution`.

Name: `hotHaloColdModeCoreRadiiMethod`
Attached to: `module:Hot_Halo_Cold_Mode_Density_Core_Radii`
File: `hot_halo.cold_mode.density_profile.core_radius.F90`
Default value: virialFraction
Description: The method to be used for `hotHaloColdModeCoreRadii`.

Name: `hotHaloMassDistributionCoreRadiusMethod`
Attached to: `module:Hot_Halo_Mass_Distributions_Core_Radii`
File: `hot_halo.mass_distribution.cored.core_radius.F90`
Default value: virialFraction
Description: The method to be used for `hotHaloMassDistributionCoreRadius`.

Name: `hotHaloMassDistributionMethod`
Attached to: `module:Hot_Halo_Mass_Distributions`
File: `hot_halo.mass_distribution.F90`
Default value: betaProfile
Description: The method to be used for `hotHaloMassDistribution`.

Name: hotHaloOutflowReincorporationMethod
Attached to: module:Hot_Halo_Outflows_Reincorporations
File: hot_halo.outflow_reincorporation.F90
Default value: haloDynamicalTime
Description: The method to be used for hotHaloOutflowReincorporation.

Name: hotHaloRamPressureForceMethod
Attached to: module:Hot_Halo_Ram_Pressure_Forces
File: hot_halo.ram_pressure_force.F90
Default value: font2008
Description: The method to be used for hotHaloRamPressureForce.

Name: hotHaloRamPressureStrippingMethod
Attached to: module:Hot_Halo_Ram_Pressure_Stripping
File: hot_halo.ram_pressure_stripping.F90
Default value: font2008
Description: The method to be used for hotHaloRamPressureStripping.

Name: hotHaloRamPressureTimescaleMethod
Attached to: module:Hot_Halo_Ram_Pressure_Stripping_Timescales
File: hot_halo.ram_pressure_stripping.timescale.F90
Default value: ramPressureAcceleration
Description: The method to be used for hotHaloRamPressureTimescale.

Name: hotHaloTemperatureProfileMethod
Attached to: module:Hot_Halo_Temperature_Profiles
File: hot_halo.temperature_profile.F90
Default value: virial
Description: The method to be used for hotHaloTemperatureProfile.

Name: initialMassFunctionMethod
Attached to: module:Stellar_Populations_Initial_Mass_Functions
File: stellar_populations.initial_mass_functions.F90
Default value: chabrier2001
Description: The method to be used for initialMassFunction.

Name: intergalacticMediumFilteringMassMethod
Attached to: module:Intergalactic_Medium_Filtering_Masses
File: intergalactic_medium.filtering_mass.F90
Default value: gnedin2000
Description: The method to be used for intergalacticMediumFilteringMass.

Name: intergalacticMediumStateMethod
Attached to: module:Intergalactic_Medium_State
File: intergalactic_medium.state.F90
Default value: recFast
Description: The method to be used for intergalacticMediumState.

Name: linearGrowthMethod
Attached to: module:Linear_Growth
File: structure_formation.linear_growth.F90
Default value: simple
Description: The method to be used for linearGrowth.

Name: massDistributionMethod
Attached to: module:Mass_Distributions
File: mass_distributions.F90
Default value: none
Description: The method to be used for massDistribution.

Name: massFunctionIncompletenessMethod
Attached to: module:Mass_Function_Incompletenesses
File: statistics.mass_function.incompleteness.F90
Default value: complete
Description: The method to be used for massFunctionIncompleteness.

Name: mergerMassMovementsMethod
Attached to: module:Satellite_Merging_Mass_Movements
File: satellites.merging.mass_movements.F90
Default value: simple
Description: The method to be used for mergerMassMovements.

Name: mergerProgenitorPropertiesMethod
Attached to: module:Satellite_Merging_Progenitor_Properties
File: satellites.merging.progenitor_properties.F90
Default value: standard
Description: The method to be used for mergerProgenitorProperties.

Name: mergerRemnantSizeMethod
Attached to: module:Satellite_Merging_Remnant_Sizes
File: satellites.merging.remnant_sizes.F90
Default value: covington2008
Description: The method to be used for mergerRemnantSize.

Name: mergerTreeBranchingProbabilityMethod
Attached to: module:Merger_Tree_Branching
File: merger_trees.branching_probability.F90
Default value: parkinsonColeHelly
Description: The method to be used for mergerTreeBranchingProbability.

Name: mergerTreeBranchingProbabilityModifierMethod
Attached to: module:Merger_Tree_Branching_Modifiers
File: merger_trees.branching_probability.modifier.F90
Default value: identity
Description: The method to be used for mergerTreeBranchingProbabilityModifier.

Name: mergerTreeBuildMassDistributionMethod

Attached to: module:Merger_Trees_Build_Masses_Distributions
File: merger_trees.construct.build.masses.distribution.F90
Default value: haloMassFunction
Description: The method to be used for mergerTreeBuildMassDistribution.

Name: mergerTreeBuildMassesMethod
Attached to: module:Merger_Trees_Build_Masses
File: merger_trees.construct.build.masses.F90
Default value: sampledDistributionUniform
Description: The method to be used for mergerTreeBuildMasses.

Name: mergerTreeBuilderMethod
Attached to: module:Merger_Trees_Builders
File: merger_trees.construct.builder.F90
Default value: cole2000
Description: The method to be used for mergerTreeBuilder.

Name: mergerTreeConstructorMethod
Attached to: module:Merger_Tree_Construction
File: merger_trees.construct.F90
Default value: build
Description: The method to be used for mergerTreeConstructor.

Name: mergerTreeEvolveTimestepMethod
Attached to: module:Merger_Tree_Timesteps
File: merger_trees.evolve.timesteps.F90
Default value: standard
Description: The method to be used for mergerTreeEvolveTimestep.

Name: mergerTreeEvolverMethod
Attached to: module:Merger_Trees_Evolve
File: merger_trees.evolver.F90
Default value: standard
Description: The method to be used for mergerTreeEvolver.

Name: mergerTreeImporterMethod
Attached to: module:Merger_Tree_Read_Importers
File: merger_trees.construct.read.importer.F90
Default value: galacticus
Description: The method to be used for mergerTreeImporter.

Name: mergerTreeMassResolutionMethod
Attached to: module:Merger_Trees_Build_Mass_Resolution
File: merger_trees.construct.build.mass_resolution.F90
Default value: fixed
Description: The method to be used for mergerTreeMassResolution.

Name: mergerTreeNodeEvolverMethod
Attached to: module:Merger_Trees_Evolve_Node

File: merger_trees.node_evolver.F90
Default value: standard
Description: The method to be used for mergerTreeNodeEvolver.

Name: mergerTreeNodeMergerMethod
Attached to: module:Merger_Trees_Merge_Node
File: merger_trees.node_merger.F90
Default value: singleLevelHierarchy
Description: The method to be used for mergerTreeNodeMerger.

Name: mergerTreeOperatorMethod
Attached to: module:Merger_Tree_Operators
File: merger_trees.operators.F90
Default value: null
Description: The method to be used for mergerTreeOperator.

Name: mergerTreeOutputterMethod
Attached to: module:Merger_Tree_Outputters
File: merger_trees.outputter.F90
Default value: standard
Description: The method to be used for mergerTreeOutputter.

Name: mergerTreeWalkerMethod
Attached to: module:Merger_Tree_Walkers
File: merger_trees.walkers.F90
Default value: isolatedNodes
Description: The method to be used for mergerTreeWalker.

Name: metaTreeProcessingTimeMethod
Attached to: module:Meta_Tree_Compute_Times
File: meta.tree_processing_times.F90
Default value: file
Description: The method to be used for metaTreeProcessingTime.

Name: modelParameterMethod
Attached to: module:Model_Parameters
File: models.parameters.F90
Default value: active
Description: The method to be used for modelParameter.

Name: nbodyHaloMassErrorMethod
Attached to: module:Statistics_NBody_Halo_Mass_Errors
File: statistics.Nbody.halos.mass_errors.F90
Default value: null
Description: The method to be used for nbodyHaloMassError.

Name: nbodyImporterMethod
Attached to: module:NBody_Importers
File: nBody.import.F90

Default value: gadgetHDF5
Description: The method to be used for `nbodyImporter`.

Name: `nbodyOperatorMethod`
Attached to: `module:NBody_Operators`
File: `nBody.operator.F90`
Default value: null
Description: The method to be used for `nbodyOperator`.

Name: `nodePropertyExtractorMethod`
Attached to: `module:Node_Property_Extractors`
File: `nodes.property_extractor.F90`
Default value: nodeIndices
Description: The method to be used for `nodePropertyExtractor`.

Name: `operatorUnaryMethod`
Attached to: `module:Math_Operators_Unary`
File: `math.operators.unary.F90`
Default value: identity
Description: The method to be used for `operatorUnary`.

Name: `outputAnalysisDistributionNormalizerMethod`
Attached to: `module:Output_Analysis_Distribution_Normalizers`
File: `galacticus.output.analyses.distribution_normalizer.F90`
Default value: identity
Description: The method to be used for `outputAnalysisDistributionNormalizer`.

Name: `outputAnalysisDistributionOperatorMethod`
Attached to: `module:Output_Analysis_Distribution_Operators`
File: `galacticus.output.analyses.distribution_operator.F90`
Default value: identity
Description: The method to be used for `outputAnalysisDistributionOperator`.

Name: `outputAnalysisMethod`
Attached to: `module:Output_Analyses`
File: `galacticus.output.analyses.F90`
Default value: null
Description: The method to be used for `outputAnalysis`.

Name: `outputAnalysisMolecularRatioMethod`
Attached to: `module:Output_Analysis_Molecular_Ratios`
File: `galacticus.output.analyses.molecular_ratio.F90`
Default value: obreschkow2009
Description: The method to be used for `outputAnalysisMolecularRatio`.

Name: `outputAnalysisPropertyOperatorMethod`
Attached to: `module:Output_Analysis_Property_Operators`
File: `galacticus.output.analyses.property_operator.F90`
Default value: identity

Description: The method to be used for `outputAnalysisPropertyOperator`.

Name: `outputAnalysisWeightOperatorMethod`
Attached to: `module:Output_Analysis_Weight_Operators`
File: `galacticus.output.analyses.weight_operator.F90`
Default value: identity
Description: The method to be used for `outputAnalysisWeightOperator`.

Name: `outputTimesMethod`
Attached to: `module:Output_Times`
File: `output.times.F90`
Default value: list
Description: The method to be used for `outputTimes`.

Name: `posteriorSampleConvergenceMethod`
Attached to: `module:Posterior_Sampling_Convergence`
File: `posterior_sampling.convergence.F90`
Default value: gelmanRubin
Description: The method to be used for `posteriorSampleConvergence`.

Name: `posteriorSampleDffrntlEvltnProposalSizeMethod`
Attached to: `module:Posterior_Sample_Differential_Proposal_Size`
File: `posterior_sampling.differential_proposal_size.F90`
Default value: none
Description: The method to be used for `posteriorSampleDffrntlEvltnProposalSize`.

Name: `posteriorSampleDffrntlEvltnPrpslSzTmpExpMethod`
Attached to: `module:Posterior_Sampling_Prop_Size_Temp_Exp`
File: `posterior_sampling.differential_proposal_size.temperature_exponent.F90`
Default value: none
Description: The method to be used for `posteriorSampleDffrntlEvltnPrpslSzTmpExp`.

Name: `posteriorSampleDffrntlEvltnRandomJumpMethod`
Attached to: `module:Posterior_Sample_Differential_Random_Jump`
File: `posterior_sampling.differential_random_jump.F90`
Default value: none
Description: The method to be used for `posteriorSampleDffrntlEvltnRandomJump`.

Name: `posteriorSampleLikelihoodMethod`
Attached to: `module:Models_Likelihoods`
File: `models.likelihoods.F90`
Default value: none
Description: The method to be used for `posteriorSampleLikelihood`.

Name: `posteriorSampleSimulationMethod`
Attached to: `module:Posterior_Sampling_Simulation`
File: `posterior_sampling.simulation.F90`
Default value: differentialEvolution
Description: The method to be used for `posteriorSampleSimulation`.

Name: posteriorSampleStateInitializeMethod
Attached to: module:Posterior_Sampling_State_Initialize
File: posterior_sampling.state.initialize.F90
Default value: priorRandom
Description: The method to be used for posteriorSampleStateInitialize.

Name: posteriorSampleStateMethod
Attached to: module:Posterior_Sampling_State
File: posterior_sampling.state.F90
Default value: simple
Description: The method to be used for posteriorSampleState.

Name: posteriorSampleStoppingCriterionMethod
Attached to: module:Posterior_Sampling_Stopping_Criteria
File: posterior_sampling.stopping_criteria.F90
Default value: never
Description: The method to be used for posteriorSampleStoppingCriterion.

Name: powerSpectrumMethod
Attached to: module:Power_Spectra
File: structure_formation.power_spectrum.F90
Default value: standard
Description: The method to be used for powerSpectrum.

Name: powerSpectrumNonlinearMethod
Attached to: module:Power_Spectra_Nonlinear
File: structure_formation.power_spectrum.nonlinear.F90
Default value: cosmicEmu
Description: The method to be used for powerSpectrumNonlinear.

Name: powerSpectrumPrimordialMethod
Attached to: module:Power_Spectra_Primordial
File: structure_formation.power_spectrum.primordial.F90
Default value: powerLaw
Description: The method to be used for powerSpectrumPrimordial.

Name: powerSpectrumPrimordialTransferredMethod
Attached to: module:Power_Spectra_Primordial_Transferred
File: structure_formation.power_spectrum.primordial.transferred.F90
Default value: simple
Description: The method to be used for powerSpectrumPrimordialTransferred.

Name: powerSpectrumWindowFunctionMethod
Attached to: module:Power_Spectrum_Window_Functions
File: structure_formation.power_spectrum.variance.window_function.F90
Default value: topHat
Description: The method to be used for powerSpectrumWindowFunction.

Name: radiationFieldMethod
Attached to: module:Radiation_Fields
File: radiation.F90
Default value: null
Description: The method to be used for radiationField.

Name: ramPressureStrippingDisksMethod
Attached to: module:Ram_Pressure_Stripping_Mass_Loss_Rate_Disks
File: ram_pressure_stripping.mass_loss_rate.disks.F90
Default value: null
Description: The method to be used for ramPressureStrippingDisks.

Name: ramPressureStrippingSpheroidsMethod
Attached to: module:Ram_Pressure_Stripping_Mass_Loss_Rate_Spheroids
File: ram_pressure_stripping.mass_loss_rate.spheroids.F90
Default value: null
Description: The method to be used for ramPressureStrippingSpheroids.

Name: satelliteDynamicalFrictionMethod
Attached to: module:Satellite_Dynamical_Friction
File: satellites.dynamical_friction.acceleration.F90
Default value: chandrasekhar1943
Description: The method to be used for satelliteDynamicalFriction.

Name: satelliteMergingTimescalesMethod
Attached to: module:Satellite_Merging_Timescales
File: satellites.merging.timescale.F90
Default value: jiang2008
Description: The method to be used for satelliteMergingTimescales.

Name: satelliteOrphanDistributionMethod
Attached to: module:Satellite_Oprhan_Distributions
File: satellites.orphans.distributions.F90
Default value: traceDarkMatter
Description: The method to be used for satelliteOrphanDistribution.

Name: satelliteTidalFieldMethod
Attached to: module:Satellites_Tidal_Fields
File: satellites.tidal_fields.F90
Default value: null
Description: The method to be used for satelliteTidalField.

Name: satelliteTidalHeatingRateMethod
Attached to: module:Satellite_Tidal_Heating
File: satellites.tidal_heating.rate.F90
Default value: zero
Description: The method to be used for satelliteTidalHeatingRate.

Name: satelliteTidalStrippingMethod

Attached to: module:Satellite_Tidal_Stripping
File: satellites.tidal_stripping.rate.F90
Default value: zentner2005
Description: The method to be used for satelliteTidalStripping.

Name: starFormationExpulsiveFeedbackDisksMethod
Attached to: module:Star_Formation_Feedback_Expulsion_Disks
File: star_formation.feedback_expulsion.disks.F90
Default value: zero
Description: The method to be used for starFormationExpulsiveFeedbackDisks.

Name: starFormationExpulsiveFeedbackSpheroidsMethod
Attached to: module:Star_Formation_Feedback_Expulsion_Spheroids
File: star_formation.feedback_expulsion.spheroids.F90
Default value: zero
Description: The method to be used for starFormationExpulsiveFeedbackSpheroids.

Name: starFormationFeedbackDisksMethod
Attached to: module:Star_Formation_Feedback_Disks
File: star_formation.feedback.disks.F90
Default value: powerLaw
Description: The method to be used for starFormationFeedbackDisks.

Name: starFormationFeedbackSpheroidsMethod
Attached to: module:Star_Formation_Feedback_Spheroids
File: star_formation.feedback.spheroids.F90
Default value: powerLaw
Description: The method to be used for starFormationFeedbackSpheroids.

Name: starFormationHistoryMethod
Attached to: module:Star_Formation_Histories
File: star_formation.histories.F90
Default value: null
Description: The method to be used for starFormationHistory.

Name: starFormationRateSurfaceDensityDisksMethod
Attached to: module:Star_Formation_Rate_Surface_Density_Disks
File: star_formation.rate_surface_density.disks.F90
Default value: krumholz2009
Description: The method to be used for starFormationRateSurfaceDensityDisks.

Name: starFormationTimescaleDisksMethod
Attached to: module:Star_Formation_Timescales_Disks
File: star_formation.timescales.disks.F90
Default value: intgrtdSurfaceDensity
Description: The method to be used for starFormationTimescaleDisks.

Name: starFormationTimescaleSpheroidsMethod
Attached to: module:Star_Formation_Timescales_Spheroids

File: star_formation.timescales.spheroids.F90
Default value: dynamicalTime
Description: The method to be used for starFormationTimescaleSpheroids.

Name: stellarAstrophysicsMethod
Attached to: module:Stellar_Astrophysics
File: stellar_astrophysics.F90
Default value: file
Description: The method to be used for stellarAstrophysics.

Name: stellarFeedbackMethod
Attached to: module:Stellar_Feedback
File: stellar_astrophysics.feedback.F90
Default value: standard
Description: The method to be used for stellarFeedback.

Name: stellarPopulationMethod
Attached to: module:Stellar_Populations
File: stellar_populations.F90
Default value: standard
Description: The method to be used for stellarPopulation.

Name: stellarPopulationPropertiesMethod
Attached to: module:Stellar_Population_Properties
File: stellar_populations.properties.F90
Default value: instantaneous
Description: The method to be used for stellarPopulationProperties.

Name: stellarPopulationSelectorMethod
Attached to: module:Stellar_Population_Selectors
File: stellar_populations.selectors.F90
Default value: fixed
Description: The method to be used for stellarPopulationSelector.

Name: stellarPopulationSpectraMethod
Attached to: module:Stellar_Population_Spectra
File: stellar_populations.spectra.F90
Default value: FSPS
Description: The method to be used for stellarPopulationSpectra.

Name: stellarPopulationSpectraPostprocessorBuilderMethod
Attached to: module:Stellar_Population_Spectra_Postprocess
File: stellar_populations.spectra.postprocess.F90
Default value: lookup
Description: The method to be used for stellarPopulationSpectraPostprocessorBuilder.

Name: stellarPopulationSpectraPostprocessorMethod
Attached to: module:Stellar_Population_Spectra_Postprocess
File: stellar_populations.spectra.postprocess.F90

Default value: inoue2014
Description: The method to be used for `stellarPopulationSpectraPostprocessor`.

Name: `stellarSpectraDustAttenuationMethod`
Attached to: `module:Stellar_Spectra_Dust_Attenuations`
File: `stellar_populations.spectra.dust_attenuation.F90`
Default value: zero
Description: The method to be used for `stellarSpectraDustAttenuation`.

Name: `stellarTracksMethod`
Attached to: `module:Stellar_Astrophysics_Tracks`
File: `stellar_astrophysics.tracks.F90`
Default value: file
Description: The method to be used for `stellarTracks`.

Name: `stellarWindsMethod`
Attached to: `module:Stellar_Astrophysics_Winds`
File: `stellar_astrophysics.winds.F90`
Default value: leitherer1992
Description: The method to be used for `stellarWinds`.

Name: `supernovaePopulationIIIMethod`
Attached to: `module:Supernovae_Population_III`
File: `stellar_astrophysics.supernovae_PopulationIII.F90`
Default value: hegerWoosley2002
Description: The method to be used for `supernovaePopulationIII`.

Name: `supernovaeTypeIaMethod`
Attached to: `module:Supernovae_Type_Ia`
File: `stellar_astrophysics.supernovae_type_Ia.F90`
Default value: nagashima2005
Description: The method to be used for `supernovaeTypeIa`.

Name: `surveyGeometryMethod`
Attached to: `module:Geometry_Surveys`
File: `geometry.surveys.F90`
Default value: liWhite2009SDSS
Description: The method to be used for `surveyGeometry`.

Name: `taskMethod`
Attached to: `module:Tasks`
File: `tasks.F90`
Default value: evolveForests
Description: The method to be used for `task`.

Name: `tidalStrippingDisksMethod`
Attached to: `module:Tidal_Stripping_Mass_Loss_Rate_Disks`
File: `tidal_stripping.mass_loss_rate.disks.F90`
Default value: null

Description: The method to be used for `tidalStrippingDisks`.

Name: `tidalStrippingSpheroidsMethod`
Attached to: `module:Tidal_Stripping_Mass_Loss_Rate_Spheroids`
File: `tidal_stripping.mass_loss_rate.spheroids.F90`
Default value: null
Description: The method to be used for `tidalStrippingSpheroids`.

Name: `transferFunctionMethod`
Attached to: `module:Transfer_Functions`
File: `structure_formation.transfer_function.F90`
Default value: eisensteinHu1999
Description: The method to be used for `transferFunction`.

Name: `unevolvedSubhaloMassFunctionMethod`
Attached to: `module:Unevolved_Subhalo_Mass_Functions`
File: `structure_formation.unevolved_subhalo_mass_function.F90`
Default value: giocoli2008
Description: The method to be used for `unevolvedSubhaloMassFunction`.

Name: `universeOperatorMethod`
Attached to: `module:Universe_Operators`
File: `universe.operators.F90`
Default value: identity
Description: The method to be used for `universeOperator`.

Name: `virialDensityContrastMethod`
Attached to: `module:Virial_Density_Contrast`
File: `structure_formation.virial_density_contrast.F90`
Default value: sphericalCollapseMatterLambda
Description: The method to be used for `virialDensityContrast`.

Name: `virialOrbitMethod`
Attached to: `module:Virial_Orbits`
File: `satellites.merging.virial_orbits.F90`
Default value: benson2005
Description: The method to be used for `virialOrbit`.

4. Extracting and Analyzing Results

Galacticus stores its output in an HDF5 file. The contents of this file can be viewed and manipulated using a variety of ways including:

HDFView This is a graphical viewer for exploring the contents of HDF5 files;

HDF5 Command Line Tools A set of tools which can be used to extract data from HDF5 files (`h5dump` and `h5ls` are particularly useful);

C++ and Fortran 90 APIs Allow access to and manipulation of data in HDF5 files;

h5py A Python interface to HDF5 files.

In the remainder of this section the structure of Galacticus HDF5 files is described and a general-purpose Perl module which we use to extract data in a convenient manner is outlined.

4.1. General Structure of Output File

Figure 4.1 shows the structure of a typical Galacticus output file. The various groups and subgroups are described below.

4.1.1. UUID

The UUID (Universally Unique Identifier) is a unique identifier assigned to each Galacticus model that is run. It allows identification of a given model and can be referenced from, for example, an external database.

4.1.2. Build Information

Galacticus automatically stores various information about how it was built in the `Build` group attributes. Currently, included attributes consist of:

`FGSL_library_version` The version number of the FGSL library;

`FoX_library_version` The version number of the FoX library;

`GSL_library_version` The version number of the GSL library;

`HDF5_library_version` The version number of the HDF5 library;

`make_CCOMPILER` The C compiler command used;

`make_CCOMPILER_VERSION` The C compiler version information;

`make_CFLAGS` The flags passed to the C compiler;

`make_CPPCOMPILER` The C++ compiler command used;

```
outputFile.hdf5
 |
 +-> UUID                                  Attribute {1}
 |
 +-> Build                                 Group
 |    |
 |    +-> FGSL_library_version             Attribute {1}
 |    +-> FoX_library_version              Attribute {1}
 |    +-> GSL_library_version              Attribute {1}
 |    +-> HDF5_library_version             Attribute {1}
 |    +-> make_CCOMPILER                   Attribute {1}
 |    +-> make_CCOMPILER_VERSION           Attribute {1}
 |    +-> make_CFLAGS                      Attribute {1}
 |    +-> make_CPPCOMPILER                 Attribute {1}
 |    +-> make_CPPCOMPILER_VERSION         Attribute {1}
 |    +-> make_CPPFLAGS                    Attribute {1}
 |    +-> make_FCCOMPILER                  Attribute {1}
 |    +-> make_FCCOMPILER_VERSION          Attribute {1}
 |    +-> make_FCFLAGS                     Attribute {1}
 |    +-> make_FCFLAGS_NOOPT               Attribute {1}
 |    +-> make_MODULETYPE                  Attribute {1}
 |    +-> make_PREPROCESSOR                Attribute {1}
 |    +-> sourceChangeSetDiff              Dataset   {1}
 |    +-> sourceChangeSetMerge             Dataset   {1}
 |
 +-> Outputs                               Group
 |    |
 |    +-> Output1                          Group
 |    |    |
 |    |    +-> nodeData                    Group
 |    |    |     |
 |    |    |     +-> nodeProperty1         Dataset {<nodeCount>}
 |    |    |     +-> ...                   Dataset {<nodeCount>}
 |    |    |     +-> ...                   Dataset {<nodeCount>}
 |    |    |     +-> ...                   Dataset {<nodeCount>}
 |    |    |     +-> nodePropertyN         Dataset {<nodeCount>}
 |    |    |
 |    |    +-> mergerTreeCount             Dataset {<treeCount>}
 |    |    |
 |    |    +-> mergerTreeIndex             Dataset {<treeCount>}
 |    |    |
 |    |    +-> mergerTreeStartIndex        Dataset {<treeCount>}
 |    |    |
 |    |    +-> mergerTreeWeight            Dataset {<treeCount>}
 |    |    |
 |    |    +-> mergerTree1                 Group             [optional]
 |    |    |     |
 |    |    |     +-> nodeProperty1         Reference
 |    |    |     +-> ...                   Reference
 |    |    |     +-> ...                   Reference
 |    |    |     +-> ...                   Reference
 |    |    |     +-> nodePropertyN         Reference
 |    |    |
 |    |    x-> ...                         Group             [optional]
 |    |    x-> ...                         Group             [optional]
 |    |    x-> ...                         Group             [optional]
 |    |    x-> mergerTree<treeCount>       Group             [optional]
 |    |    |
```

`make_CPPCOMPILER_VERSION` The C++ compiler version information;

`make_CPPFLAGS` The flags passed to the C++ compiler;

`make_FCCOMPILER` The Fortran compiler command used;

`make_FCCOMPILER_VERSION` The Fortran compiler version information;

`make_FCFLAGS` The flags passed to the Fortran compiler;

`make_FCFLAGS_NOOPT` The flags passed to the Fortran compiler for unoptimized compiles;

`make_MODULETYPE` The Fortran module type identifier string;

`make_PREPROCESSOR` The preprocessor command used.

Additionally, two datasets are included which store details of the Galacticus source changeset. `sourceChangeSetMerge` contains the output of "`hg bundle -t none`", that is, it contains a Mercurial changegroup that incorporates any changes made to the current branch relative to the main Galacticus branch. `sourceChangeSetDiff` contains the output of "`hg diff`", that is, all differences between the source code in the working directory and that which has been committed to Mercurial. Used together, these two datasets allow the precise source code used to run the model to be recovered from the main branch Galacticus source.

4.1.3. Filters

For each broadband filter used in the Galacticus model run an entry is added to the datasets in this group. Currently, two datasets are generated:

`name` The name of each filter used.

`wavelengthEffective` The effective wavelength, $\lambda_{\rm eff}$ (defined as $\lambda_{\rm eff} = \int_0^\infty \lambda R(\lambda) {\rm d}\lambda / \int_0^\infty R(\lambda) {\rm d}\lambda$, where $R(\lambda)$ is the filter response) of the filter in Å.

4.1.4. Parameters

The `Parameters` group contains a record of all parameter values (either input or default) that were used for this Galacticus run. The group contains a long list of attributes, each attribute named for the corresponding parameter and with a single entry giving the value of that parameter. If a parameter has subparameters, a group is created having the same name as the parameter, which will contain attributes corresponding to each subparameter. The `scripts/aux/Extract_Parameter_File.pl` script can be used to extract these parameter values to an XML file suitable for re-input into Galacticus.

4.1.5. Version

The `Version` group contains a record of the Galacticus version used for this model, storing the major and minor version numbers, the revision number and the Mercurial revision and hash (if the code is being maintained using Mercurial, otherwise a value of -1 is entered or the revision and the hash attribute is empty). Additionally, the time at which the model was run is stored and, if the `galacticusConfig.xml` file (see §2.1) is present and contains contact details, the name and e-mail address of the person who ran the model.

4.1.6. globalHistory

The `globalHistory` group stores volume averaged properties of the model universe as a function of time. Currently, the properties stored are:

`historyTime` Cosmic time (in Gyr);

`historyExpansion` Expansion factor;

`historyStarFormationRate` Volume averaged star formation rate (in $M_\odot$/Gyr/Mpc^3).

`historyDiskStarFormationRate` Volume averaged star formation rate in disks (in $M_\odot$/Gyr/Mpc^3).

`historySpheroidStarFormationRate` Volume averaged star formation rate in spheroids (in $M_\odot$/Gyr/Mpc^3).

`historyStellarDensity` Volume averaged stellar mass density (in $M_\odot$/Mpc^3).

`historyDiskStellarDensity` Volume averaged stellar mass density in disks (in $M_\odot$/Mpc^3).

`historySpheroidStellarDensity` Volume averaged stellar mass density in spheroids (in $M_\odot$/Mpc^3).

`historyGasDensity` Volume averaged cooled gas density (in $M_\odot$/Mpc^3).

`historyNodeDensity` Volume averaged resolved node density (in $M_\odot$/Mpc^3).

Dimensionful datasets have a `unitsInSI` attribute which gives their units in the SI system.

4.1.7. Outputs

The `Outputs` group contains one or more sub-groups corresponding to the output times requested from Galacticus. Each sub-group contains the following information:

`outputTime` ***(attribute)*** The cosmic time (in Gyr) at this output;

`outputExpansionFactor` ***(attribute)*** The expansion factor at this output;

`nodeData` A group of node properties as described below.

`mergerTree` **subgroups** ***(optional)*** A set of `mergerTree` groups as described below.

Output is controlled by parameters given within the `mergerTreeOutput` section of the parameter file. Current options are:

`outputMergerTrees` If `true` then each merger tree is output to the relevant sub-group at each output time (see §4.1.7). Otherwise merger trees are not output. [Default: `true`.]

`outputReferences` If `true` then an HDF5 reference dataset is written for each merger tree subgroup (see §4.1.7). [Default: `false`.]

`galacticFilterMethod` A "galactic filter" (see §16.4.1) which is applied to each node in the tree to determine whether or not it should be output. By combinding multiple filters it is possible to construct arbitrarily complex criteria for output. [Default: `always`.]

nodeData group

The `nodeData` group contains all data from nodes in all merger trees. The group consists of a collection of datasets each of which lists a property of all nodes in the trees which exist at the output time. Where relevant, each dataset contains an attribute, `unitsInSI`, which gives the units of the dataset in the SI system.

mergerTree datasets

To allow locating of nodes belonging to a given merger tree in the datasets in the `nodeData` group, the `mergerTreeStartIndex` and `mergerTreeCount` datasets list the starting index of each tree's nodes in the `nodeData` datasets, and the number of nodes belonging to each tree respectively. Additionally, the `mergerTreeWeight` dataset lists the `volumeWeight` property for each tree (see §4.1.7) which gives the weight (in Mpc^{-3}) which should be assigned to this tree (and all nodes in it) to create a volume-averaged sample (see §4.2.1). Finally, the `mergerTreeIndex` dataset gives the index of each tree stored in the `nodeData` datasets.

mergerTree subgroups

These subgroups will be present if the `[mergerTreeOutputReferences]` parameter is set to true. Each `mergerTree` subgroup contains HDF5 references to all data on a single merger tree. The group consists of a collection of scalar references each of which points to the appropriate region of the corresponding dataset in the `nodeData` group. Additionally, the `volumeWeight` attribute of this group gives the weight (in Mpc^{-3}) which should be assigned to this tree (and all nodes in it) to create a volume-averaged sample. (A second attribute, `volumeWeightUnitsInSI`, gives the units of `volumeWeight` in the SI system.)

4.1.8. Optional Outputs

Numerous other quantities can be optionally output. These are documented below:

Redshifts

The redshift corresponding to the time at which a node was last isolated can be output by setting `[outputNodeRedshifts]` to `true`. This quantity will be output as `basicRedshiftLastIsolated`.

Mass Accretion Histories

A mass accretion history (i.e. mass as a function of time) for the main branch in each merger tree can be output by setting `massAccretionHistoryOutput=true`. If requested, a new group `massAccretionHistories` will be made in the Galacticus output file. It will contain groups called `mergerTreeN` where `N` is the merger tree index. Each such group will contain the following three datasets, defined for the main branch of the tree[1]:

`nodeIndex` The index of the node in the tree;

`nodeTime` The time at this point in the tree (in Gyr);

`nodeMass` The mass of the node at this point in the tree (in $M_\odot$). The `nodeMass` property is defined to be the total mass of each node in a merger tree. Therefore, it includes both dark and baryonic mass. Additionally, the mass of a node includes the mass of any satellite nodes that it may contain. The mean density of the node depends on the method selected by the `virialDensityContrastMethod` parameter.

[1] "Main branch" is defined by starting from the root node of a tree and repeatedly stepping back to the most massive progenitor of the branch. This does not necessarily pick out the most massive progenitor at a given time.

Merger Tree Dump

A full dump of merger tree structure by setting `mergerTreeStructureDump=true`. In this case, files will be dumped to the directory specified by `[mergerTreeStructureDumpDirectory]` for each merger tree with final mass between `[mergerTreeStructureDumpMassMinimum]` and `[mergerTreeStructureDumpMassMaximum]`. Each tree is dumped to a file named "`mergerTreeDump:<treeIndex>:1.gv`" in the specified directory in GraphViz format.

Conditional Mass Functions

Setting `[mergerTreeComputeConditionalMassFunction]=true` will cause conditional mass functions to be computed and output to the Galacticus output file in a group named "`conditionalMassFunction`". The mass functions are binned in parent halo mass, and the mass ratio of the progenitor to parent halo. Bins are logarithmically spaced in mass (and mass ratio), with the range and number of bins controlled by the parameters:

- `[mergerTreeComputeConditionalMassFunctionParentMassCount]`;
- `[mergerTreeComputeConditionalMassFunctionParentMassMinimum]`;
- `[mergerTreeComputeConditionalMassFunctionParentMassMaximum]`;
- `[mergerTreeComputeConditionalMassFunctionMassRatioCount]`;
- `[mergerTreeComputeConditionalMassFunctionMassRatioMinimum]`;
- `[mergerTreeComputeConditionalMassFunctionMassRatioMaximum]`.

The resulting parent masses and mass ratios are written to datasets `massParent` and `massRatio` respectively. Parent and progenitor halos are defined at a set of redshifts defined by the arrays `[mergerTreeComputeConditionalMas` and `[mergerTreeComputeConditionalMassFunctionProgenitorRedshifts]`, which are written to datasets `redshiftParent` and `redshiftProgenitor`. The resulting conditional masses functions are written to datasets `conditionalMassFunction` and `conditionalMassFunctionError`.

In addition to standard progenitor mass functions, the progenitor mass function conditioned on progenitor rank (i.e. $1^{\rm st}$ most massive, $2^{\rm nd}$, $\ldots$, $n^{\rm th}$ most massive progenitor) is computed and output to the datasets `primaryProgenitorMassFunction` and `primaryProgenitorMassFunctionError`. The depth (i.e. n) is specifed by `[mergerTreeComputeConditionalMassFunctionPrimaryProgenitorDepth]`.

Finally, the progenitor mass functoin conditioned on recent formation is computed and output to the datasets `formationRateFunction` and `formationRateFunctionError`. To be considered "recently formed" a progenitor must have formed between t and $t(1-\Delta)$ where t is the progenitor time and $\Delta=$`[mergerTreeConditionalMassFunctionFormationRateTimeFraction]`.

Pre-Evolution Merger Trees

Galacticus can output the full structure of merger trees prior to any evolution. Merger tree structure can be requested by setting `mergerTreeStructureOutput=true`. Structures are written to a new group, `mergerTreeStructures`, in the Galacticus output file. This group will contain groups called `mergerTreeN` where `N` is the merger tree index. Each such group will contain the following datasets:

`nodeIndex` The index of the node in the tree;

`childIndex` The index of this node's first child node;

`parentIndex` The index of this node's parent node;

`siblingIndex` The index of this node's sibling node;

`nodeTime` The time at this point in the tree (in Gyr);

`nodeMass` The mass of the node at this point in the tree (in $M_\odot$). The `nodeMass` property is defined to be the total mass of each node in a merger tree. Therefore, it includes both dark and baryonic mass. Additionally, the mass of a node includes the mass of any satellite nodes that it may contain. The mean density of the node depends on the method selected by the `virialDensityContrastMethod` parameter.

Additional, optional, datasets can be added by setting appropriate input parameters. Currently these include:

Virial quantities If `mergerTreeStructureOutputVirialQuantities=true` then two additional datasets are included:

`nodeVirialRadius` The virial radius of the node (in Mpc);

`nodeVirialVelocity` The virial velocity of the node (in km/s);

ark matter scale radii If `mergerTreeStructureOutputDarkMatterScaleRadius=true` then an additional dataset is included:

`darkMatterScaleRadius` The scale radius of this node's dark matter halo profile (in Mpc);

tree final descendent If `outputFinalDescendentIndices=true` then an additional dataset is included:

`finalDescendentIndex` The index of the final descendent that this node will reach in its merger trees;

4.2. Topics in Analysis of GALACTICUS Outputs

4.2.1. Building Volume Limited Samples

The `mergerTreeWeight` property (see §4.1.7) property specifies the weight to be assigned to each merger tree in a model to construct a representative (i.e. volume limited) sample of galaxies. GALACTICUS does not typically generate every merger tree in a fixed volume of the Universe (as an N-body simulation might for example) as it's generally a waste of time to simulate millions of low mass halos and only a small number of high mass halos. The `mergerTreeWeight` factors correct for this sampling. If merger trees are being built, then the `mergerTreeWeight`, w_i, for each tree of mass M_i (where the trees are ranked in order of increasing mass) is given by

$$w_i = \int_{M_{\rm min}}^{M_{\rm max}} n(M) {\rm d}M, \tag{4.1}$$

where $n(M)$ is the dark matter halo mass function and

$$M_{\rm min} = \sqrt{M_{i-1} M_i}, \tag{4.2}$$
$$M_{\rm min} = \sqrt{M_i M_{i+1}}. \tag{4.3}$$

Suppose, for example, that we wish to construct a luminosity function of galaxies. In particular, we consider a luminosity bin k which extends from $L_k - \Delta k/2$ to $L_k + \Delta k/2$. If tree i contains N_i galaxies with luminosities $l_{i,j}$, where j runs from 1 to N_i, then the luminosity function in this bin is given by:

$$\phi_k = \sum_i \sum_{j=1}^{N_i} \left\{ \begin{array}{ll} w_i & \hbox{if } L_k - \Delta k/2 < l_{i,j} \leq L_k + \Delta k/2 \\ 0 & \hbox{otherwise.} \end{array} \right. \tag{4.4}$$

4.3. Meta-Analysis of Galacticus

Galacticus contains modules which allow it to analyze and profile its own performance.

4.3.1. Tree Construction/Evolution Timing

The Galacticus_Meta_Tree_Timing module records the time taken to construct and evolve each merger tree. Setting [metaCollectTimingData]=true will cause tree timing data to be recorded and output to the metaData/treeTiming group. Three datasets are written to this group:

treeMasses Gives the base node masses of the recorded trees (in units of $M_\odot$);

treeConstuctTimes Gives the time (in seconds) taken to construct each merger tree;

treeEvolveTimes Gives the time (in seconds) taken to evolve each merger tree.

4.3.2. ODE Evolver Profiler

The Galacticus_Meta_Evolver_Profiler module records statistics on the performance of the main ODE solver used to advance galaxies through time.

Note: Currently, this profiler requires access to features of the GNU Scientific Library that are not implemeted within FGSL. As such, this functionallity is normally *not* compiled with Galacticus. If you want to use the profiler, contact Andrew Benson and request a copy of the modified FGSL source code. Once this is installed, the profiler can be activated by including -DPROFILE in the compilation options (e.g. add this to your GALACTICUS_FCFLAGS environment variable).

When active, setting [profileOdeEvolver]=true will activate profiling. Each step taken by the ODE evolver is then analyzed. First, a record of the size of the time step taken is recorded. Second, the property which is currently limiting the time step size (i.e. that which has the largest error over the step as judged using the same heuristics as the ODE solver uses to determine step size) is determined and a record of this is kept.

At the end of a run the accumulated data is written to the Galacticus output file, into a group named metaData/evolverProfiler. A histogram of time step sizes is written to metaProfileTimeStepCount with bins specified in metaProfileTimeStep—these bins can be adjusted using [metaProfileTimeStepMinimum], [metaProfileTimeStepMaximum] and [metaProfileTimeStepPointsPerDecade]. A histogram of which properties limited step size is written to metaProfilePropertyHitCount with the associated property names written to [metaProfilePropertyNames]. Property names can only be determined if the component to which they belong supports the decodePropertyIdentifiersTask directive (see §16.4.3). Properties which could not be decoded in this way are listed as unknown.

4.4. On-The-Fly Analysis

Galacticus can perform various analyses on-the-fly (i.e. as it is running), outputting the expectation for a particular observed dataset. Analyses are selected by adding the appropriate label to the [mergerTreeAnalyses] parameter (multiple, space-separated labels can be added to this parameter). Available on-the-fly analyses are described below.

On-the-fly analyses exist for several different stellar and gas mass functions. These analyses share several common features which are described below. In general, the model masses are used to construct a mass function by binning into a histogram using the same bins as the observational dataset as the centers of the bins (with bin boundaries placed at the geometric means of consecutive bin centers), and with each galaxy contributing a weight equal to the volume weight of its merger tree. Typically the bins

will be uniformly spaced in the logarithm of mass. Before accumulation to the histogram, galaxy masses are adjusted to account for two effects:

1. *Cosmological parameters:* Typically the observational data will have been analyzed assuming some specific set of cosmological parameters which will differ from that in the current model. Therefore, the mass of the galaxy and the weight assigned to it are both adjusted to match what would be inferred if they were assessed using the same cosmological parameters as were used for the observational data. Typically, this will mean that masses are scaled in proportion to $D_\mathrm{L}^2(\bar{z})$, where $D_\mathrm{L}(z)$ is the luminosity distance to redshift z and $\bar{z}$ is the median redshift of observational sample, while weights are scaled in proportional to $D_\mathrm{c}^2 H^{-1}(z)$, where $D_\mathrm{c}(z)$ is the comoving distance to redshift z and $H(z)$ is the Hubble parameter at redshift z. These scalings are typical for galaxies where masses are determined from luminosities, but may vary in other cases.

2. *Systematic errors:* Each mass function allows for a systematic shift in model masses (to account for systematic uncertainties in the observational analysis) using a simple model. Specifically, model masses are mapped by this model as follows

$$\log_{10} M \rightarrow \log_{10} M + \sum_{i=0}^{N} \alpha_i \log_{10}^i (M/M_0), \tag{4.5}$$

   where M_0 is a mass scale defined for each analysis, N is fixed integer for each analysis, and the coefficients α_i are input parameters `[<label>MassSystematic<i>]`, where `<label>` is the label of the analysis and `<i>` is an integer in the range $0 \ldots N$.

Individual analyses may defined additional transformations of the galaxy mass. These are detailed below.

When the weight of each galaxy is accumulated to the mass function histogram, the logarithm of the galaxy mass is modeled as a Gaussian kernel with width specified by each analysis to account for random errors in the observations and/or scatter in model masses. That is, the weight of each galaxy is spread over every bin of the histogram using this Gaussian kernel. Additionally, if `[analysisMassFunctionsApplyGravitationalLensing` then the mass of each galaxy is convolved with the magnification distribution expected due to gravitational lensing from large-scale structure (see §12.19).

The contribution to the mass function from each model output redshift is computed from the volume associated with that output redshift, given the angular geometry and depth of the survey as determined from the appropriate survey geometry (see §12.59) and assuming that each output redshift represents a range of redshifts running between the geometric means of the times corresponding to each output redshift.

In addition to the mass function, the covariance matrix, $\mathbf{C}_\mathrm{model}$, of the mass function is also computed. The assumptions used when constructing the covariance matrix are controlled by the parameter `[analysisMassFunctionCovarianceModel]`. If set to `binomial`, them to construct $\mathbf{C}_\mathrm{model}$ we make use of the fact that Galacticus works by sampling a set of tree "root masses" from the $z = 0$ dark matter halo mass function. From each root, a tree is grown, within which the physics of galaxy formation is then solved. Root masses are sampled uniformly from the halo mass function. That is, the cumulative halo mass function, $N(M)$, is constructed between the maximum and minimum halo masses to be simulated. The number of root masses, N_r, to be used in a model evaluation is then determined. Root masses are then chosen such that

$$N(M_i) = N(M_\mathrm{min}) \frac{i-1}{N_\mathrm{r}-1} \tag{4.6}$$

for $i = 1 \ldots N_\mathrm{r}$ (noting that $N(M_\mathrm{max}) = 0$ by construction).

Consider first those galaxies which form in the main branch of each tree (i.e. those galaxies which are destined to become the central galaxy of the $z = 0$ halo). Suppose that we simulate N_k halos of root mass M_k at $z = 0$. In such halos the main branch galaxies will, at any time, have stellar masses drawn

from some distribution $p_k(M_\star|t)$. The number of such galaxies contributing to bin i of the mass function is therefore binomially distributed with success probability $p_{ik} = \int_{M_{i,min}}^{M_{i,max}} p_k(M_\star|t)\mathrm{d}M_\star$ and a sample size of N_k. The contribution to the covariance matrix from these main branch galaxies is therefore:

$$\mathcal{C}_{ij} = \begin{cases} p_{ik}(1-p_{ik})N_k w_k^2 & \text{if } i = j \\ -p_{ik}p_{jk}N_k w_k^2 & \text{otherwise,} \end{cases} \tag{4.7}$$

where w_k is the weight to be assigned to each tree. To compute this covariance requires knowledge of the probabilities, p_{ik}. We estimate these directly from the model. To do this, we bin trees into narrow bins of root mass and assume that p_{ik} does not vary significantly across the mass range of each bin. Using all realizations of trees that fall within a given bin, k, we can directly estimate p_{ik}. In computing p_{ik}, the range of halo masses considered and the fineness of binning in halo mass are determined by the parameters `[analysisMassFunctionsHaloMassMinimum]`, `[analysisMassFunctionsHaloMassMaximum]`, and `[analysisMassFunctionsHaloMassBinsPerDecade]`.

If instead, `[analysisMassFunctionCovarianceModel]`=`Poisson`, the main branch galaxies are modeled as being sampled from a Poisson distribution (and so off-diagonal terms in the covariance matrix will be zero).

In addition to the main branch galaxies, each tree will contain a number of other galaxies (these will be "satellite" galaxies at $z = 0$, but at higher redshifts may still be central galaxies in their own halos). Tests have established that the number of satellites in halos is well described by a Poisson process. Note that, as described above, each galaxy contributes a Gaussian distribution to the mass function due to modelling of random errors in stellar mass determinations. For main branch galaxies this is simply accounted for when accumulating the probabilities, p_{ik}. For satellite galaxies, off-diagonal contributions to the covariance matrix arise as a result, $C_{ij} = w_k f_i f_j$, where f_i is the fraction of the galaxy contributing to bin i of the mass function.

The parameter `[analysisMassFunctionsCorrelationTruncateLevel]` allows off-diagonal elements in the model covariance matrix whose correlation is less than the specified value to be truncated to zero. This helps avoid remove numerical noise from the covariance matrix.

4.4.1. ALFALFA HI Mass Function

This analysis, selected using label `alfalfaHiMassFunctionZ0.00`, computes the model expectation for the HI gas mass function of Martin et al. [2010]. Calculation of the mass function follows the basic methdology outlined above. Model galaxy masses and weights are mapped for differences in cosmological parameters as described above, and the simple systematic mass error model is employed with parameter $N = 1$ and $M_0 = 10^9 M_\odot$.

The analysis assumes that only the total interstellar medium (ISM) mass of each galaxy is available, along with the disk radius (assuming an exponential disk). To infer the HI mass the model of Obreschkow et al. [2009] is used. Specifically, the molecular ratio, $R_{\rm mol} \equiv M_{\rm H_2}/M_{\rm HI}$, is given by:

$$R_{\rm mol} = \left(A_1 R_{\rm mol}^{{\rm c}\,\alpha_1} + A_2 R_{\rm mol}^{{\rm c}\,\alpha_2}\right)^{-1}, \tag{4.8}$$

where the ratio at the disk center is given by

$$R_{\rm mol}^{\rm c} = [K r_{\rm disk}^{-4} M_{\rm gas}(M_{\rm gas} + \langle f_\sigma \rangle M_\star)]^\beta. \tag{4.9}$$

Here, $R_{\rm mol}$ is the mass ratio of H_2 to HI, $M_\star$ is the stellar mass of the disk, $r_{\rm disk}$ is the disk exponential scale length, $\langle f_\sigma \rangle$ is the average ratio of the vertical velocity dispersions of gas to stars, and $K = {\rm G}/(8\pi P_\star)$. The HI mass is then determined from:

$$M_{\rm HI} = X_{\rm H} M_{\rm gas}/(1 + R_{\rm mol}), \tag{4.10}$$

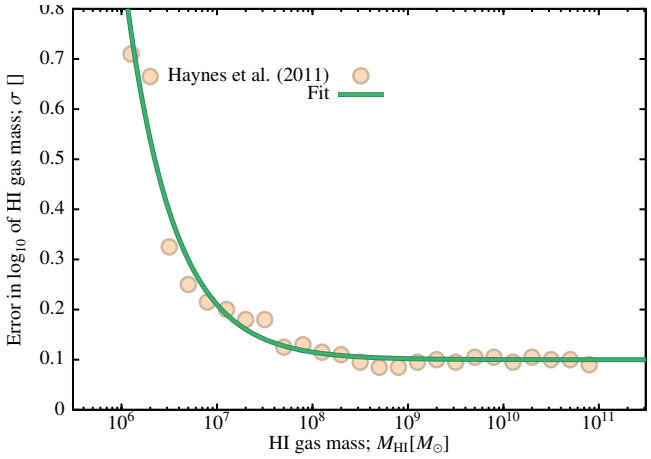

Figure 4.2.: The observational random error in galaxy HI mass as a function of HI mass for the ALFALFA survey. Points show the errors reported by Haynes et al. [2011], while the line shows a simple functional form fit to these errors.

where $X_{\rm H} = 0.778$ is the primordial hydrogen fraction by mass. In the above $K =$ `[alfalfaHiMassFunctionZ0.00MolecularF`
$\langle f_\sigma \rangle =$ `[alfalfaHiMassFunctionZ0.00MolecularFractionfSigma]`, $A_1 =$ `[alfalfaHiMassFunctionZ0.00MolecularFracti`
$A_2 =$ `[alfalfaHiMassFunctionZ0.00MolecularFractionA2]`, $\alpha_1 =$ `[alfalfaHiMassFunctionZ0.00MolecularFractionAlp`
$\alpha_2 =$ `[alfalfaHiMassFunctionZ0.00MolecularFractionAlpha2]`, and $\beta =$ `[alfalfaHiMassFunctionZ0.00MolecularFract`
Default values for these parameters are taken from Obreschkow et al. [2009]. According to Obreschkow (private communication), there remains significant scatter of $\sigma_{R_{\rm mol}} = 0.4$ dex between the predicted $R_{\rm mol}$ from this model and that observed. This is accounted for in when constructing the mass function (see below).

To account for both observational errors and scatter in $R_{\rm mol}$ not captured by the above model, the HI mass of each galaxy is modeled as a Gaussian in $\log_{10} M_{\rm HI}$ when constructing the mass function. Observational random errors on HI mass, including those arising from flux density uncertainties and errors in the assumed distance to each source, are taken from Fig. 19 of Haynes et al. [2011]. The magnitude of the error as a function of HI mass is fit using a functional form:

$$\sigma_{\rm obs} = a + \exp\left(-\frac{\log_{10}(M_{\rm HI}/M_\odot) - b}{c}\right), \tag{4.11}$$

where $\sigma_{\rm obs}$ is the error on $\log_{10}(M_{\rm HI}/M_\odot)$. We find a reasonable fit using values[2] of $a = 0.100 \pm 0.010$, $b = 5.885 \pm 0.100$, and $c = 0.505 \pm 0.020$ as shown in Fig. 4.4.1. The total random error on the logarithm of each galaxy mass is given by $\sigma^2 = \sigma^2_{R_{\rm mol}} + \sigma^2_{\rm obs}$, and is used as the width of the Gaussian kernal when applying each galaxy to the mass function histogram (as described above).

Additionally, HI mass estimates can be affected by HI self-absorption for highly inclined galaxies. [Zwaan et al., 1997, see also Zwaan et al. 2005] estimate that this effect would lead to a mean underestimation of HI masses by a factor 1.1 for a randomly oriented galaxy sample. Therefore, a value of -0.0414 for the systematic parameter `[alfalfaHiMassFunctionZ0.00MassSystematic0]` is recommended.

[2]This should not be regarded as a formal good fit. Error estimates are approximate—we have simply found a functional form that roughly describes them, along with conservative errors on the parameters of this function which are included in the priors.

5. Input Data

In some configurations, Galacticus requires additional input data to run. For example, if asked to process galaxy formation through a set of externally derived merger trees, then a file describing those trees must be given. In the remainder of this section we describe the structure of external datasets which can be inputs to Galacticus.

5.1. Broadband Filters

To compute luminosities through a given filter, Galacticus requires the response function, $R(\lambda)$, of that filter to be defined. Galacticus follows the convention of Hogg et al. [2002] in defining the filter response to be the fraction of incident photons received by the detector at a given wavelength, multiplied by the relative photon response (which will be 1 for a photon-counting detector such as a CCD, or proportional to the photon energy for a bolometer/calorimeter type detector. Filter response files are stored in `data/filters/`. Their structure is shown below, with the `SDSS_g.xml` filter reponse file used as an example:

```
 <filter>
  <description>SDSS g vacuum (filter+CCD +0 air mass)</description>
  <name>SDSS g</name>
  <origin>Michael Blanton</origin>
  <response>
    <datum>   3630.000      0.0000000E+00</datum>
    <datum>   3680.000      2.2690000E-03</datum>
    <datum>   3730.000      5.4120002E-03</datum>
    <datum>   3780.000      9.8719997E-03</datum>
    <datum>   3830.000      2.9449999E-02</datum>
    .
    .
    .
  </response>
  <effectiveWavelength>4727.02994472695</effectiveWavelength>
  <vegaOffset>0.107430167298754</vegaOffset>
</filter>
```

The `description` tag should provide a description of the filter, while the `name` tag provides a shorter name. The `origin` tag should describe from where/whom this filter originated. The `response` element contains a list of `datum` tags each giving a wavelength (in Angstroms) and response pair. The normalization of the response is arbitrary. The `effectiveWavelength` tag gives the mean, response-weighted wavelength of the filter and is used, for example, in dust attenuation calculations. The `vegaOffset` tag gives the value (in magnitudes) which must be added to an AB-system magnitude in this system to place it into the Vega system. Both `effectiveWavelength` and `vegaOffset` can be computed by running

```
scripts/filters/vega_offset_effective_lambda.pl data/filters
```

which will compute these values for any filter files that do not already contain them and append them to the files.

5.2. Merger Trees

While Galacticus can build merger trees using analytic methods it is often useful to be able to utilize merger trees from other sources (e.g. extracted from an N-body simulation). To facilitate this, Galacticus allows merger trees to be read from an HDF5 file[1]. To do so, set the `[mergerTreeConstructMethod]` input parameter to `read` and specify the filename to read via their `[mergerTreeReadFileName]` parameter.

The HDF5 file should follow the general purpose format described in §A. An example of how to construct such a file can be found in the `tests/nBodyMergerTrees` folder. In that folder, the `getMillenniumTrees.pl` script will retrieve a sample of merger trees from the Millennium Simulation database and use the `Merger_Tree_File_Maker.exe` code supplied with Galacticus to convert these into an HDF5 file suitable for reading into Galacticus. The `getMillenniumTrees.pl` script requires you to have a username and password to access the Millennium Simulation database[2]. These can be entered manually or stored in a section of the `galacticusConfig.xml` file (see §2.1) as follows:

```
<millenniumDB>
  <host>
    <name>^myHost$</name>
    <user>myUserName</user>
    <passwordFrom>input</passwordFrom>
    <treePath>/path/to/trees</treePath>
  </host>
  <host>
    <name>default</name>
    <user>myUserName</user>
    <password>myPassword</password>
  </host>
</millenniumDB>
```

Here, each `host` section describes rules for a given computer (with "default" being used if no specific match to the regular expression give in `name` is found). The `user` element gives the user name to use, while the `passwordFrom` element specifies how the password should be obtained. Currently the only allowed mechanism is "input", in which case the password is read from standard input. Alternatively, you can include a `password` element which contains the password itself. Of course, this is insecure...

The optional `treePath` element gives the location where merger trees from the Millennium Simulation can be stored. Some scripts will make use of this location so that Millennium Simulation merger trees can be shared between multiple scripts.

5.2.1. Processing of Merger Tree Files

The "read" merger tree construction method (see §12.32.1) reads these files and processes them into a form suitable for Galacticus to evolve. Merger trees are inherently complex structures, particularly when the possibility of subhalos are considered. Galacticus is currently designed to work with single descendent merger trees, i.e. ones in which the tree structure is entirely defined by specifying which node a given node is physically associated with at a later time. Additionally, Galacticus expects the merger tree file to contain information on the host node, i.e. the node within which a given node is physically

[1] The following assumes that merger trees will be read from a file following Galacticus's standard HDF5 format which is described in §A. Other formats can also be read by selecting the relevant importer via the `[mergerTreeImporterMethod]` parameter.

[2] If you do not have a username and password for the Millennium Simulation database you can request one from contact@g-vo.org.

located. In the following, these two properties are labelled `descendentNode` and `hostNode`. Galacticus assumes that nodes for which `descendentNode`=`hostNode` are isolated halos (i.e. they are their own hosts) while other nodes are subhalos (i.e. they are hosted by some other node). An example of a simple tree structure is shown in Fig. 5.1. The particular structure would be represented by the following list of nodes and node properties (a -1 indicates that no descendent node exists):

node	descendentNode	hostNode
1	-1	1
2	1	2
3	2	3
4	1	4
5	4	5
6	-1	1
7	6	4
8	7	8

The following should be noted when constructing merger tree files:

- Note that Galacticus does not require that nodes be placed on a uniform grid of times/redshifts, nor that mass be conserved along a branch of the tree. After processing the tree in this way, Galacticus builds additional links which identify the child node of each halo and any sibling nodes. These are not required to specify the tree structure but are computationally convenient.
- It is acceptable for a node to begin its existence as a subhalo (i.e. to have never had an isolated node progenitor). Such nodes will be created as satellites in the merger tree and, providing the selected node components (see §11) initialize their properties appropriately, will be evolved correctly.
- It is acceptable for an isolated node to have progenitors, none of which are a primary progenitor. This can happen if all progenitors descend into subhalos in the isolated node. In such cases, Galacticus will create a clone of the isolated node at a very slightly earlier time to act as the primary progenitor. This is necessary to allow the tree to be processed correctly, but does not affect the evolution of the tree.
- Normally, cases where a node's host node cannot be found in the forest will cause Galacticus to exit with an error. Setting `[mergerTreeReadMissingHostsAreFatal]`=`false` will instead circumvent this issue by making any such nodes self-hosting (i.e. they become isolated nodes rather than subhalos). Note that this behavior is not a physically correct way to treat such cases—it is intended only to allow trees to be processed in cases where the full forest is not available.
- It is acceptable for nodes to jump between branches in a tree, or even to jump between branches in different trees. In the latter case, all trees linked by jumping nodes (a so-called "forest" of connected trees) must be stored as a single forest (with multiple root-nodes) in the merger tree file. Galacticus will process this forest of trees simultaneously, allowing to nodes to move between their branches.
- It is acceptable for a subhalo to later become an isolated halo (as can happen due to three-body interactions; see Sales et al. 2007). If `[mergerTreeReadAllowSubhaloPromotions]`=`true` then such cases will be handled correctly (i.e. the subhalo will be promoted back to being an isolated halo). If `[mergerTreeReadAllowSubhaloPromotions]`=`false` then subhalos are not permitted to become isolated halos. In this case, the following logic will be applied to remove all such cases from the tree:

  → For any branch in a tree which at some point is a subhalo:

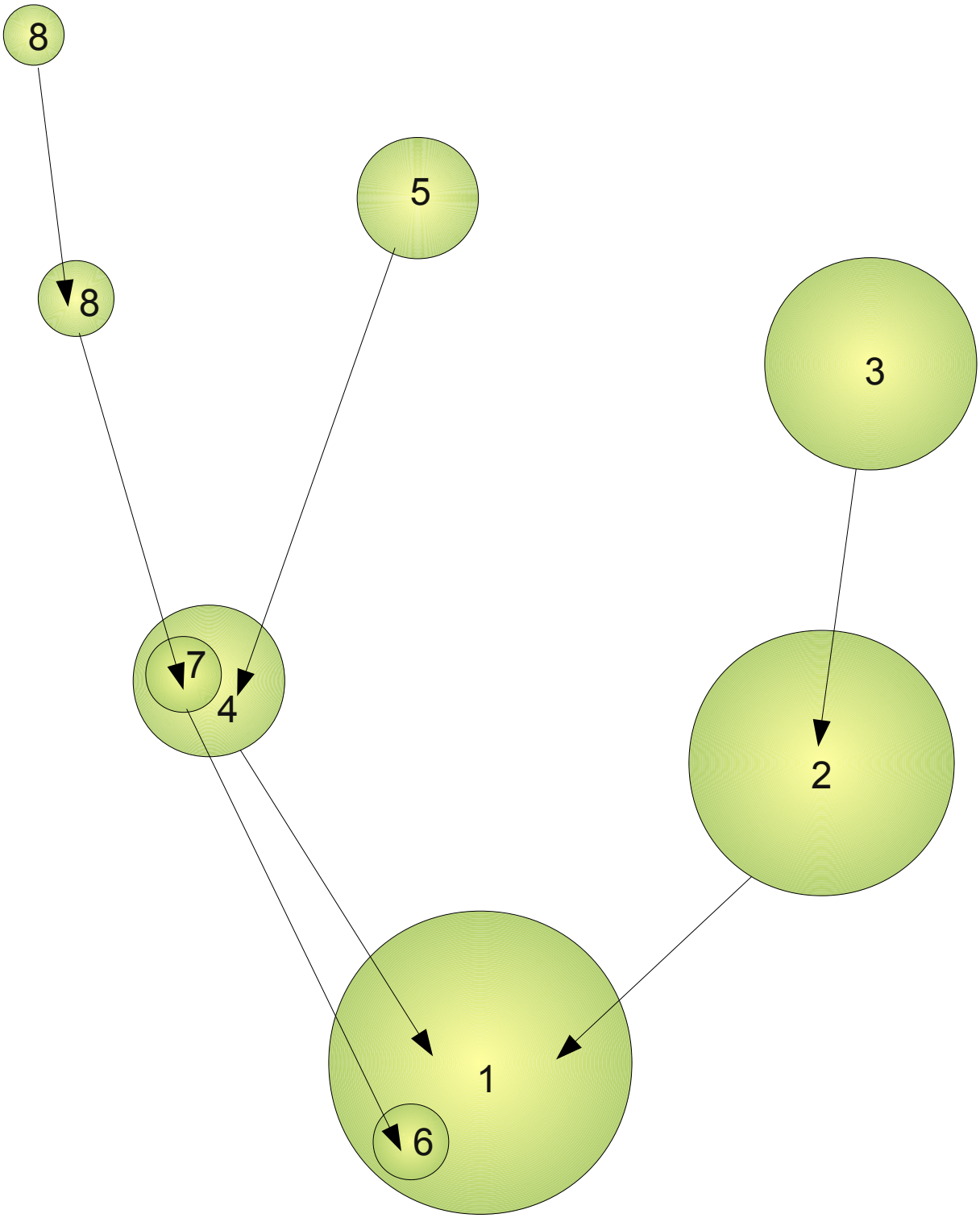

Figure 5.1.: An example of a simple merger tree structure. Colored circles represent nodes in the merger tree. Each node has a unique index indicated by the number inside each circle. Black arrows link each node to its descendent node (as specified by the `descendentNode` property. Where a node is not its own host node it is placed inside its host node.

→ Beginning from the earliest node in the branch that is a subhalo, repeatedly step to the next descendent node;

→ If that descendent is *not* a subhalo then:

→ If there is not currently any non-subhalo node which has the present node as its descendent then current node is only descendent of a subhalo. Therefore, try to make this node a subhalo, and propose the descendent of the host node of the previous node visited in the branch as the new host:

→ If the proposed host exists:

→ If the mass of the current node is less than that of the proposed host:

→ If the proposed hosts exists before the current node, repeatedly step to its descendents until one is found which exists at or after the time of the current node. This is the new proposed host.

→ If the proposed host is a subhalo, make it an isolated node.

→ The current node is made a subhalo within the proposed host.

→ Otherwise:

→ The current node remains an isolated node, while the proposed host is instead made a subhalo within the current node.

→ Otherwise:

→ The proposed host does not exists, which implies the end of a branch has been reached. Therefore, flag the current node as being a subhalo with a host identical to that of the node from which it descended.

Requirements for Galacticus Input Parameters

The following requirements must be met for the input parameters to Galacticus when using merger trees read from file:

- The cosmological parameters (Ω_M, Ω_Λ, Ω_b, H_0, σ_8), if defined in the file, must be set identically in the Galacticus input file unless you set `[mergerTreeReadMismatchIsFatal]=false` in which case you'll just be warned about any mismatch;
- Galacticus assumes by default that all merger trees exist at the final output time—if this is not the case set `[allTreesExistAtFinalTime]=false`.

5.2.2. Setting of Halo Properties

Dark Matter Scale Radii

If `[mergerTreeReadPresetScaleRadii]=true` and the `halfMassRadius` dataset is available within the `haloTrees` group (see §A.7) then the half-mass radii of nodes will be used to compute the corresponding

scale length of the dark matter halo profile[3]. This requires a dark matter profile scale component which supports setting of the scale length (see §11.13).

Satellite Merger Times

If `[mergerTreeReadPresetMergerTimes]=true` then merger times for satellites will be computed directly from the merger tree data read from file. When a subhalo has an isolated halo as a descendent it is assumed to undergo a merger with that isolated halo at that time. Note that this requires a satellite orbit component method which supports setting of merger times (e.g. `[treeNodeMethodSatelliteOrbit]=preset`).

Dark Matter Halo Spins

If `[mergerTreeReadPresetSpins]=true` and the `angularMomentum` dataset is available within the `haloTrees` group (see §A.7) then the spin parameters of nodes will be computed and set. This requires a dark matter halo spin component which supports setting of the spin (see §11.12).

[3]The scale radius is found by seeking a value which gives the correct half mass radius. It is therefore important that the definition of halo mass (specifically the virial overdensity) in Galacticus be the same as was used in computing the input half mass radii.

6. Tutorials

This chapter contains step-by-step guides to performing common tasks with Galacticus.

6.1. Running Galacticus on N-body Merger Trees

See §A.11 for details of how to build merger tree files suitable for input into Galacticus. There are many options which control precisely how merger trees read from file should be handled. The following section provides guidance on the best choice of parameters.

6.1.1. Setting Input Parameters

To utilize merger trees from the file[1] that you created in a Galacticus run it's necessary to set two parameters in the input parameter file that you will use for the run:

```
<!-- Specify that merger trees are to be read from file, and give the name of the file to read -->
<mergerTreeConstructMethod value="read"             />
<mergerTreeReadFileName    value="myNBodyTrees.hdf5"/>
```

The first of these `[mergerTreeConstructMethod]=read` tells Galacticus that merger trees will be constructed by reading them from a file. The second, `[mergerTreeReadFileName]`, gives the name of the file from which to read the trees.

In order to choose sensible settings for the various parameters that control merger trees read from file, it is recommended that you read through each of the items below and follow the guidance given.

Cosmology: In addition to specifying that trees should be read from a file, it's also important to ensure that the values of cosmological parameters in Galacticus match those in the merger tree file. (If they don't match, Galacticus will stop with an error message unless you set `[mergerTreeReadMismatchIsFatal]=false` in which case you'll just be warned about any mismatch.) In our case of using merger trees from the Millennium Simulation, the correct cosmological parameter values can be set as follows:

```
<!-- Use Millennium Simulation cosmology. -->
<cosmologyParametersMethod value="simple"/>
 <HubbleConstant  value="73.0"  />
 <OmegaMatter     value="0.25"  />
 <OmegaDarkEnergy value="0.75"  />
 <OmegaBaryon     value="0.0455"/>
</cosmologyParametersMethod>
<cosmologicalMassVarianceMethod value="filteredPower">
  <sigma_8 value="0.900"/>
</cosmologicalMassVarianceMethod>
<powerSpectrumPrimordialMethod value="powerLaw">
  <index               value="1.000"/>
  <wavenumberReference value="1.000"/>
```

[1] The following assumes that merger trees will be read from a file following Galacticus's standard HDF5 format which is described in Appendix A.

```
    <running              value="0.000"/>
  </powerSpectrumPrimordialMethod>
```

Existance at Final Time: Normally, Galacticus assumes that all merger trees will exist (i.e. have at least one node present) at the final output time. This may not be true of trees extracted from an N-body simulation—in this case Galacticus can be informed of this fact by setting:

```
<allTreesExistAtFinalTime value="false"/>
```

Snapping Nodes to Snapshots: N-body merger trees are often built from "snapshots" of the simulation, i.e. all of the nodes exist at a set of discrete times. Often we want to output nodes at precisely these output times. In such cases it is useful to set:

```
<mergerTreeReadOutputTimeSnapTolerance value="1.0d-3"/>
```

which ensures that the times of nodes are adjusted to lie at precisely the output time if that time is within the specified relative tolerance (this avoids any small differences between node times and output times that can arises due to rounding errors when converting from redshifts to times and vice-versa).

Missing Hosts: Galacticus expects to find each node's host node present in a merger tree forest. If a node's host is not found this is cause for a fatal error to be issued, since it is impossible to correctly construct and evolve the corresponding forest. If you absolutely want to run a forest for which one or more host nodes are missing, you can allow this by setting `[mergerTreeReadMissingHostsAreFatal]=false`—in this case missing host nodes trigger a warning only and nodes without a host are forced to become isolated nodes. This will lead to incorrect tree evolution however, so the recommended setting is:

```
<mergerTreeReadMissingHostsAreFatal value="true"/>
```

Branch Jumps and Subhalo Promotions: If your merger trees contain subhalos they will most likely exhibit two specific behaviors[2]: i) nodes which are subhalos in one timestep may become non-subhalos (isolated halos) in a subsequent timestep ("subhalo promotion"), and ii) nodes which are subhalos in one branch of the tree may "jump" to another branch[3] of the tree becoming a subhalo there ("branch jumping"). These behaviors are fully supported by Galacticus and so we recommend the following settings:

```
<mergerTreeReadAllowSubhaloPromotions value="true"/>
<mergerTreeReadAllowBranchJumps       value="true"/>
```

You may choose to disallow these behaviors by setting either of the above parameters to `false`—for example if you wish to explore how your results would differ if subhalos were forced to remain subhalos forever in their original branch. Note that allowing subhalo promotion while not allowing branch jumping can lead to deadlocks in merger tree evolution, so change these settings with caution.

Note that for trees which do not contain subhalos these two parameters are irrelevant.

Subhalo Masses: If your trees contain subhalos, the mass evolution of those subhalos can be preset in the satellite component of each node. In this way, the subhalo mass in Galacticus will track that specified by the merger tree file. This requires the use of a satellite component which allows presetting of subhalo masses. Recommended settings are therefore:

```
<treeNodeMethodSatellite           value="preset"/>
<mergerTreeReadPresetSubhaloMasses value="true"  />
```

If your trees do not contain subhalos, recommended settings are instead:

[2] These two behaviors are called out as they specifically *do not* occur in merger trees created using Press-Schechter-based algorithms for example.

[3] That is, the subhalo's descendented is hosted by a node other than the descendent of the subhalo's host.

```
<treeNodeMethodSatellite           value="standard"/>
<mergerTreeReadPresetSubhaloMasses value="false"   />
```

Halo Positions/Velocities: If your trees contain position and velocity information for halos, those positions and velocities can be preset in the position component of each node. This requires the use of a position component which allows presetting of positions and velocities. Recommended settings are therefore:

```
<treeNodeMethodPosition        value="preset"/>
<mergerTreeReadPresetPositions value="true"  />
```

If your trees do not contain position information recommended settings are:

```
<treeNodeMethodPosition        value="null" />
<mergerTreeReadPresetPositions value="false"/>
```

Subhalo Orbits: If your trees contain position and velocity information they can be used to preset initial orbit information for subhalos. Note that it is not required that your trees contain subhalos for this orbit presetting to be performed—Galacticus can follow subhalo orbits even if subhalos are not included in the trees themselves. The following settings are recommended:

```
<mergerTreeReadPresetOrbits             value="true"/>
<mergerTreeReadPresetOrbitsSetAll       value="true"/>
<mergerTreeReadPresetOrbitsAssertAllSet value="true"/>
<mergerTreeReadPresetOrbitsBoundOnly    value="true"/>
```

These options will cause an orbit to be preset for each subhalo based on the relative position and velocity of merging halos and assuming that the orbital energy and angular momentum are conserved between the time immediately prior to the merger and the time of virial radius crossing. If the computed orbit does not cross the virial radius of the larger halo or if the computed orbit is unbound, the above options cause an orbit to be preset by drawing orbital parameters at random from the chosen cosmological distribution (see §12.51.2).

Subhalo Merging: If your merger trees contain subhalo information, that information can be used to specify when, and with which other node, each subhalo merges. Specifically, a subhalo is assumed to merge at the time at which it is not the primary progenitor of its descendent halo—possibly with some other delay to be described below. Recommended settings are:

```
<mergerTreeReadPresetMergerTimes          value="true"             />
<mergerTreeReadPresetMergerNodes          value="true"             />
<mergerTreeReadSubresolutionMergingMethod value="boylanKolchin2008"/>
```

The first two options cause subhalos to merge at the time described above, and with their descendent node. The final option accounts for the possibility that the subhalo should not actually merge immediately at this time. For example, in N-body simulations, the subhalo may have simply been lost due to limitations of resolution or halo finder algorithms. The final option specifies that some additional time until merging be added based on the subhalo merging timescale algorithm of Boylan-Kolchin et al. [2008; see §12.51.1], and computed using the last known orbital properties of the subhalo.

Halo Scale Radii: If your merger trees contain information on halo scale radii or half-mass radii, these can be used to preset the scale radius of each node. This requires the use of a dark matter profile component which allows presetting of scale length. Recommended settings are therefore:

```
<mergerTreeReadPresetScaleRadii                  value="true"     />
<mergerTreeReadPresetScaleRadiiFailureIsFatal    value="true"     />
```

```
<mergerTreeReadPresetScaleRadiiConcentrationMinimum value="3"        />
<mergerTreeReadPresetScaleRadiiConcentrationMaximum value="60"       />
<mergerTreeReadPresetScaleRadiiMinimumMass          value="see below"/>
```

Minimum and maximum concentrations are specified—these are used to restrict the range of scale radii that are allowed for a given halo. If scale radii are to be determined based on half-mass radii given in the merger tree file, and if the computed scale radius does not result in a concentration between these limits, then a fatal error is issued.

Finally, you can set a minimum halo mass via the `[mergerTreeReadPresetScaleRadiiMinimumMass]` parameter below which the scale radii or half-mass radii in your file should be considered not reliable. For halos below this mass, scale radii will instead be assigned via the selected dark matter halo concentration method (see §12.11.3).

Halo Angular Momenta: If your merger trees contain spin or angular momentum information these can be preset for each node. Recommended settings are:

```
<treeNodeMethodSpin                   value="preset"/>
<mergerTreeReadPresetSpins            value="true"  />
<mergerTreeReadPresetUnphysicalSpins value="true"  />
```

The last of these options causes any halos for which the spin given in the merger tree file is non-positive to be assigned a spin at random instead, drawn from the specified cosmological distribution (see §12.11.7).

If subhalo masses are not included in their host halo masses in your merger tree file, you should specify how the angular momenta of subhalos should be accounted for when adding their mass to their host halo. If positions and 3D angular momenta are available in your merger tree file, the recommended setting is:

```
<mergerTreeReadSubhaloAngularMomentaMethod value="summation"/>
```

If this information is not present

```
<mergerTreeReadSubhaloAngularMomentaMethod value="scale"    />
```

should be used instead.

If your merger tree file contains 3D spin or angular momentum information, you can choose to make that information available within Galacticus be using the settings:

```
<treeNodeMethodSpin          value="preset3D"/>
<mergerTreeReadPresetSpins3D value="true"    />
```

Subhalo Indices: If your merger trees contain subhalos, you can tell Galacticus to keep track of the indices of subhalos by setting:

```
<treeNodeMethodSatellite            value="preset"/>
<mergerTreeReadPresetSubhaloIndices value="true"  />
```

The Galacticus output file will then contain `satelliteNodeIndex` datasets which list the index (as given in the merger tree file) for all subhalos and halos. Without specifying this presetting, the index of subhalos is frozen at the index of the halo immediately prior to it becomming a subhalo.

The remainder of this section gives more detail about many of the parameters described above and how they affect handling of merger trees read from file. Further parameters can be set to control what information from the stored trees will be used in Galacticus. Examples are given below.

6.1.2. Further Details

Further details of the effects of the many parameters controlling merger trees read from file are given below.

Node Positions

If position and velocity information for tree nodes is available within the merger tree file then Galacticus can be instructed to use this information by using the "preset" method for tree node positions and telling the merger tree construction method to preset node positions as follows:

```
<!-- Use merger tree node positions -->
<treeNodeMethodPosition        value="preset"/>
<mergerTreeReadPresetPositions value="true"  />
```

If position information is unavailable, the "null" position method can be selected and the merger tree construction method instructed not to preset positions as follows:

```
<!-- Do not use merger tree node positions -->
<treeNodeMethodPosition        value="null" />
<mergerTreeReadPresetPositions value="false"/>
```

Virial Orbits

If position and velocity information for tree nodes is available within the merger tree file then Galacticus can be instructed to use this information to estimate the orbit of each subhalo at the point at which it crosses the virial radius of its host halo. This "virial orbit" may then be used by, for example, calculations of merging timescales.

```
<!-- Use merger tree node positions to compute orbits at the virial radius -->
<mergerTreeReadPresetOrbits             value="true"/>
<mergerTreeReadPresetOrbitsBoundOnly    value="true"/>
<mergerTreeReadPresetOrbitsSetAll       value="true"/>
<mergerTreeReadPresetOrbitsAssertAllSet value="true"/>
```

Typically, a merging halo is not seen at precisely the time at which it crosses the virial radius of its host (due to the fact that N-body simulations are output at discretely spaced timesteps). Therefore, Galacticus computes the orbit at the time just prior to merging and assumes that the orbital parameters (energy and angular momentum) remain fixed to propagate the orbit to the virial radius of the host. The second parameter in the above example, `[mergerTreeReadPresetOrbitsBoundOnly]`, specifies whether or not only bound orbits should be set. Some calculations (e.g. of subhalo merging times) assume bound orbits and may fail if given an unbound orbit. Setting this option to `true` causes only bound orbits to be preset—unbound orbits are ignored. Note that some orbits cannot be propagated to the virial radius (i.e. their pericenter is larger than the virial radius). The `[mergerTreeReadPresetOrbitsSetAll]` option, if true, will cause such orbits to be assigned randomly using the selected `[virialOrbitsMethod]`, such that all orbits are assigned. The `[mergerTreeReadPresetOrbitsAssertAllSet]` option requires that all orbits be set—if `[mergerTreeReadPresetOrbitsSetAll]`=`false` and `[mergerTreeReadPresetOrbitsAssertAllSet]`=`true` then Galacticus will exit with an error message if any orbit cannot be set.

If the satellite component additionally permits setting of the satellite position and velocity, these properties will also be assigned based on the relative position and velocity of the satellite and host halos.

Merging Times and Targets

The times at which subhalos merge with their host halo can be determined directly from the merger tree file if subhalo information is included in that file. Merging is assumed to occur when the subhalo no longer has a distinct descendent (i.e. it descends into a non-subhalo). If merging times are to be computed in this way set

```
<treeNodeMethodSatellite         value="preset"/>
<mergerTreeReadPresetMergerTimes value="true"  />
```

which select a satellite orbit method that allows merger times to be present and tell the merger tree construction method to preset those merger times respectively. If merger times are not to be computed in this way then instead set, for example,

```
<treeNodeMethodSatellite         value="standard" />
<mergerTreeReadPresetMergerNodes value="false"    />
<satelliteMergingMethod          value="Jiang2008"/>
```

which selects a standard satellite orbit method, prevents attempts to preset the merger times and selects the `Jiang2008` method for computing merger times instead.

In addition to setting the times of merger events, it is possible to set the target node with which a merging node should merge. By default, Galacticus will assume that all merging occurs with the non-subhalo host node in which a subhalo is located. This may not be the desired behavior when using N-body merger trees. For example, such trees may indicate that a subhalo merges with another subhalo. Setting

```
<mergerTreeReadPresetMergerNodes value="true"/>
```

will cause the target node with which each merger should occur to be determined from the merger tree structure and preset for use in Galacticus.

It is possible to add a delay between the last time at which a subhalo was seen in a simulation and the time at which it is considered to merge. This functionality is motivated by the consideration that a subhalo vanishing from a simulation may be simply due to it dropping below resolution rather than it actually having undergone a merger. The parameter `[mergerTreeReadSubresolutionMergingMethod]` can be used to select a satellite merging timescale method (see §12.51.1) to use in this case. (It is set by default to "`null`" such that no delay before merging occurs.) The orbit of the subhalo around its parent at the last time it is present in the merger tree is passed to this method and used to estimate a time until merging. This delay is added to the time at which the subhalo merges and, if merge target nodes are being set, the target node is updated accordingly.

Subhalo Indices

The indices of subhalos are usually frozen at the index of the halo just prior to becoming a subhalo. The index of the corresponding halo in the original tree (as read from file) can be tracked as follows:

```
<treeNodeMethodSatellite            value="preset"/>
<mergerTreeReadPresetSubhaloIndices value="true"  />
```

to first select the "preset" satellite orbit method (which allows subhalo indices to be preset) and, second, to instruct the merger tree construction algorithm to preset those indices. The index will then be available in output as `satelliteNodeIndex`.

Subhalo Masses

The masses of subhalos (specifically their time evolution after they become subhalos) can be set using the values stored in the merger tree file (if available). To set subhalo masses in this way use

```
<treeNodeMethodSatellite           value="preset"/>
<mergerTreeReadPresetSubhaloMasses value="true"  />
```

to first select the "preset" satellite orbit method (which allows subhalo masses to be preset) and, second, to instruct the merger tree construction algorithm to preset those masses.

Node Spins

If information on the angular momenta of nodes is available in the merger tree file, this can be used to preset the value of the spin parameter in each node[4] by setting:

```
<mergerTreeReadPresetSpins value="true"/>
```

The spin parameter is set using the spin of each node if available, or otherwise using the angular momentum of each node stored in the merger tree file using:

$$\lambda = \frac{|\mathbf{J}||E|^{1/2}}{GM^{5/2}} \tag{6.1}$$

where $|\mathbf{J}|$ is the magnitude of the node's angular momentum, M is the node's mass and E is its energy. Additionally, by setting:

```
<mergerTreeReadPresetSpins3D value="true"/>
```

the spin vector of each node will be set (assuming that the vector spin or angular momenta of nodes are available in the merger tree file) using:

$$\lambda = \frac{\mathbf{J}|E|^{1/2}}{GM^{5/2}}. \tag{6.2}$$

If spins could not be determined for some halos the spin (or angular momentum) should be set to zero in the merger tree file, and the parameter `[mergerTreeReadPresetUnphysicalSpins]` set to `true`. Galacticus will then assign a spin to such halos by sampling from the selected spin distribution (see §12.11.7).

Node Scale Radii

If information on the half-mass or scale radii of nodes is available in the merger tree file, it can be used to preset the value of the dark matter halo scale radius in each node by setting:

```
<mergerTreeReadPresetScaleRadii value="true"/>
```

Before doing this, it is important to be sure that the half-mass or scale radii of the nodes are reliable. For example, in low mass nodes extracted from an N-body simulation resolution effect may limit the accuracy of the measured half-mass or scale radius. In such cases, use the `[mergerTreeReadPresetScaleRadiiMinimumMass]` parameter to specify the lowest mass halos for which the scale radii should be preset—lower mass halos will be assigned a scale radius using the method specified by the `[mergerTreeReadConcentrationFallbackMethod]` parameter (which will default to the value of `[darkMatterConcentrationMethod]`; see §12.11.3). It is also possible to specify minimum and maximum allowed concentrations when computing the scale radius from the half mass radius using the `[mergerTreeReadPresetScaleRadiiConcentrationMinimum]` and `[mergerTreeReadPresetScaleRadiiConcentrationMaximum]` parameters. If matching the half mass radius would require a concentration outside of this range, Galacticus will abort unless `[mergerTreeReadPresetScaleRadiiFa` in which case it will instead silently use the fallback concentration method described above.

If only half-mass radii are available, the scale radius is set by using a root finding algorithm to ensure that half of the total halo mass is enclosed within the specified half-mass radius.

[4] Before doing this, it is important to be sure that the angular momenta of the nodes are reliable. For example, in low mass nodes extracted from an N-body simulation resolution effect may limit the accuracy of the measured angular momentum.

Miscellaneous N-body Properties

Several miscellaneous properties often available from N-body merger trees can also be preset by setting the following parameters to `true`:

mergerTreeReadPresetParticleCounts Sets the number of particles in each halo (requires the `particleCount` dataset to be present in the merger tree file);

mergerTreeReadPresetVelocityMaxima Sets the maxima of halo rotation curves (requires the `velocityMaximum` dataset to be present in the merger tree file);

mergerTreeReadPresetVelocityDispersions Sets the velocity dispersion of halos (requires the `velocityDispersion` dataset to be present in the merger tree file).

Subhalo Promotion

A subhalo may, at a later time, become an isolated halo once again. Galacticus allows you to control whether such behavior is allowed, or should be prohibited. To allow such "subhalo promotion", set:

```
<mergerTreeReadAllowSubhaloPromotions value="true"/>
```

If you choose to inhibit this behavior by setting the above parameter to false, a halo that becomes a subhalo will remain a subhalo forever thereafter. Note that the isolated halo to which it would have been promoted will still exist, and may therefore form its own galaxy. This can result in double counting of mass, and so inhibiting subhalo promotion is not recommended.

"Fly-by" Halos

In some cases, a halo that is part of one tree can later become part of another tree. This can happen in so-called "fly-by" encounters where a halo may briefly become a subhalo in a halo in tree A then leave that halo and become a subhalo in tree B.

The correct way to handle this issue is to combine trees A and B into a single tree (which will now have multiple base nodes). Galacticus will then process these two trees simultaneously, correctly handling the fly-by, and outputting the trees as two separate trees.

If for some reason this is not possible or desired, the fly-by problem will normally cause Galacticus to complain that the host halo of a node cannot be found (since it exists in a different tree). This problem can be avoided by setting:

```
<mergerTreeReadMissingHostsAreFatal value="false"/>
```

In this case, nodes with missing hosts are simply treated as being isolated halos. This will avoid an error condition, but is not a physically correct way to handle such cases, so use with caution.

6.1.3. Using Particles to Track Unresolved Subhalos

In N-body simulations it is possible that a subhalo can become "lost" from the simulation (i.e. can no longer be identified by a halo finder due to resolution issues) before it has actually merged with the central galaxy or been completely tidally destroyed. In such cases it is useful to be able to assign a position to the subhalo at later times. A common approach to assigning a position (and velocity) is to use that of the most bound particle in the subhalo at the last time it was identified. Galacticus allows for particle tracking in this way through the addition of particle information to the merger tree file.

To add particle tracking data to a merger tree file, follow these steps:

1. Identify all subhalos which are lost from the simulation prior to the final timestep;

2. Determine the index of the most bound particle in each such subhalo in the last timestep in which it was identified;

3. For each subhalo, extract the redshift, position, and velocity of that particle (which is usally trivial to do once its index is known) at each subsequent timestep in the simulation;

4. Write these data (along with the particle index) to the `particles` group in the merger tree file as described in §A.10;

5. Add two datasets to the `forestHalos` group:

   a) `particleIndexStart` which should indicate the index in the datasets in the `particles` group at which the data for each halo begins (or -1 if no particle data is included for the halo);

   b) `particleIndexCount` which should indicate the number of entries in the datasets in the `particles` group for each halo (or -0 if no particle data is included for the halo).

6.1.4. Handling of Extremely Large Merger Tree Forests

Halos can move between merger trees (see §6.1.2 and §6.1.2), leading to the necessity of merger tree forests—interconnected groups of merger trees that Galacticus typically processes as a whole. These forests can become very large—in some cases so large that they do not fit within the available memory. Galacticus can handle such forests by splitting them into individual trees. Each tree is processed separately, and nodes are moved between trees as needed. If a tree needs a node from another tree before its evolution can continue, its state can be suspended to disk, and later re-read once the node it requires becomes available. In this way, very large forests can be processed without running out of memory (as trees are stored to disk while they are not being processed).

To cause forests to be split in this way, the following parameters should be set:

```
<treeEvolveSuspendToRAM                   value="false"            />
<treeEvolveSuspendPath                    value="/my/scratch/path/"/>
<mergerTreeReadForestSizeMaximum          value="10000000"         />
<mergerTreeReadSubresolutionMergingMethod value="infinite"         />
```

Here, `treeEvolveSuspendToRAM` specifies that merger trees should be suspended to disk (i.e. not to RAM which is the default), and `treeEvolveSuspendPath` gives a path where the suspended trees can be written—typically scratch space local to the compute node where Galacticus is being run is a good option.

`mergerTreeReadForestSizeMaximum` specifies the maximum number of nodes allowed in a forest before it will be split. A suitable number for this depends on the details of the available RAM, the number of threads sharing that RAM, and the characteristics of the Galacticus model being used (which will affect the memory required per node).

Finally, `mergerTreeReadSubresolutionMergingMethod` is set to `infinite` to prevent any merging (which is not supported for split forests at present, although it should be soon).

6.1.5. Analyzing the Output

Positions and Velocities

Components of the position of each node are output as `positionX`, `positionY` and `positionZ` and can be accessed in the same way as other output properties from Galacticus (see §4.1.7).

Subhalo Masses

The current mass of subhalos is available via the `nodeBoundMass` output dataset and can be accessed in the same way as other output properties from Galacticus (see §4.1.7). For non-subhalos this property is equal to the usual `nodeMass` property.

6.2. Generating Mock Catalogs with Lightcones from the Millennium Simulation

Suppose that you want to create a catalog of galaxies as would be found in a survey of an area of the sky out to some redshift. Such a "mock catalog" can be built by populating with galaxies all of the dark matter halos which happen to lie within the cone which that area makes as it is projected from the observer through the Universe.

Generating such a mock catalog using Galacticus involves first extracting the halos (and their merger trees) within this "lightcone" from a suitable N-body simulation, and then processing them through Galacticus. In this tutorial, we will specifically make use of the Millennium Simulation database to provide the merger trees, but the same principles apply to any N-body simulation.

The script, `scripts/aux/Millennium_Lightcone_Grab.pl` can be used to retrieve merger trees that intersect a given lightcone from the Millennium Database and to store them in Galacticus's format (see §A). The script is used as follows:

```
scripts/aux/Millennium_Lightcone_Grab.pl <lightconeDirectory> <fieldSize> <maximumRedshift>
    --user <myUserName> --password <myPassword> --treesPerFile <treesPerFile>
```

Here, `<lightconeDirectory>` is the name of a (pre-existing) directory into which merger tree data will be stored, `<fieldSize>` is the length (in degrees) of one side of the square field of view of the lightcone, `<maximumRedshift>` is the highist redshift for which halos should be included in the catalog. The `-user` and `-password` options allow you to specify your username and password for accessing the Millennium Simulation database. Finally, the `-treesPerFile` specifies how many merger trees should be stored in each file (the script will split the lightcone between many files—this is primarily so that each request sent to the Millennium Database server is not too large). If no value is specified a default of 200 trees per file will be used.

The script generates multiple SQL queries to the Millennium database in order to first find all halos which intersect the lightcone and second to retrieve the complete merger tree associated with each such halo. These merger trees are then stored in Galacticus's merger tree file format in files named `Lightcone_Trees_AAA:BBB.hdf5` in the given `<lightconeDirectory>`, where `AAA` and `BBB` are numbers giving the first and last trees in the file[5]

Each of the merger tree files created can then be run through Galacticus in the usual way (see §6.1). Outputs should be requested at every Millennium snapshot (up to the largest redshift to be considered), and the `lightcone` filter should be used to cause only those galaxies which intersect the lightcone to be output—for example:

```
<!-- Set output redshifts to the snapshots in the milliMillennium. -->
<outputRedshifts value=
   "0.0000 0.0199 0.0414 0.0645 0.0893 0.1159 0.1444 0.1749 0.2075 0.2425
    0.2798 0.3197 0.3623 0.4079 0.4566 0.5086 0.5642 0.6236 0.6871 0.7550
    0.8277 0.9055 0.9887 1.0779 1.1734 1.2758 1.3857 1.5036 1.6303 1.7663
    1.9126 2.0700 2.2395 2.4220 2.6189 2.8312 3.0604 3.3081 3.5759 3.8657
    4.1795 4.5196 4.8884 5.2888 5.7239 6.1968"
```

[5] Note that these are not the ID numbers of the trees, just a sequential count of all trees retrieved.

```
/>

<!-- Add a lightcone filter with the required geometry -->
<mergerTreeOutput>
  <galacticFilterMethod value="lightcone"/>
</mergerTreeOutput>

<!-- Switch on output of lightcone data -->
<outputLightconeData value="true"/>

<!-- Prune away trees not appearing in the lightcone -->
<mergerTreeOperatorMethod value="pruneLightcone"/>

<!-- Specify lightcone geometry -->
<geometryLightconeMethod value="square">
  <origin value="0 0 0"/>
  <unitVector1 value=" 1 1  1"/>
  <unitVector2 value=" 0 1 -1"/>
  <unitVector3 value="-2 1  1"/>
  <lengthReplication value="500"/>
  <lengthHubbleExponent value="-1"/>
  <lengthUnitsInSI value="3.08567758135e22"/>
  <angularSize value="0.5"/>
  <timeEvolvesAlongLightcone value="true"/>
  <redshift value=
   "0.0000 0.0199 0.0414 0.0645 0.0893 0.1159 0.1444 0.1749 0.2075 0.2425
    0.2798 0.3197 0.3623 0.4079 0.4566 0.5086 0.5642 0.6236 0.6871 0.7550
    0.8277 0.9055 0.9887 1.0779 1.1734 1.2758 1.3857 1.5036 1.6303 1.7663
    1.9126 2.0700 2.2395 2.4220 2.6189 2.8312 3.0604 3.3081 3.5759 3.8657
    4.1795 4.5196 4.8884 5.2888 5.7239 6.1968"
  />
</geometryLightconeMethod>
```

In the above `[outputLightconeData]` causes lightcone coordinate information (i.e. the position and velocity of each galaxy in a coordinate system with axes aligned along the line of sight of the lightcone and parallel to the two edges of the square field of view, along with the redshift) to be output (see §13.9), and `[mergerTreeOperatorMethod]` is set to `pruneLightcone` to cause any merger trees which have no nodes within the lightcone volume to be pruned away (as there is no need to process them). Finally, the `geometryLightconeMethod` parameter describes the geometry of the lightcone to be used—see §16.4.1 for details.

6.3. Using the Instantaneous Recycling Approximation

Choosing `[stellarPopulationPropertiesMethod]=instantaneous` will cause Galacticus to use the instantaneous recycling approximation for all calculations of stellar populations. The recyling rate and yield to use are set by the `[imfNAMERecycledInstantaneous]` and `[imfNAMEYieldInstantaneous]` parameters respectively, where `NAME` is the name of the appropriate IMF.

Setting `[stellarPopulationPropertiesMethod]=noninstantaneous` causes Galacticus to use a fully non-instantaneous, metal-depdendent calculation of recycling, metal production and SNe rates.

However, it is possible to force this method to operate in the instantaneous recycling approximation limit (which can be useful for testing and comparison) by setting:

```
<!-- Force the calculation of recycling, yields etc. to   -->
<!-- be done assuming instantaneous recycling             -->
<starFormationImfInstantaneousApproximation value="true"/>
<!-- Set the mass of stars which should be used as the    -->
<!-- dividing line between long-lived and instantaneously -->
<!-- evolving in this approximation.                      -->
<starFormationImfInstantaneousApproximationMassLongLived value="1.0"/>
<!-- Set the effective age of populations to use in this -->
<!-- approximation when computing SNe numbers.           -->
<starFormationImfInstantaneousApproximationEffectiveAge value="13.8"/>
```

6.4. Postprocessing of Stellar Spectra

Stellar luminosities are computed by convolving a library of simple stellar populations with the star formation history of each galaxy. Galacticus allows the spectra of those simple stellar populations to be postprocessed (after being read from file or internally generated for example) before they are utilized in the convolution integral. This postprocessing can modify the spectra in arbitrary ways that depend on wavelength, redshift, and age of stellar population. Furthermore, Galacticus allows you to chain together stellar spectra postprocessors into a set to allow multiple postprocessings to be applied. Furthermore again, you can define an arbitrary number of sets and apply different sets to different luminosities.

Typical uses of stellar spectra postprocessors include accounting for absorption of galaxy light by the intervening IGM, or capturing only the light from recent star formation[6]. A full list of the available postprocessors can be found in §12.49.

If you don't specify a postprocessing set, the "default" set (consisting of the `inoue2014` postprocessor; see §12.49.1) is applied to each luminosity calculation. To specify other postprocessing sets add the following to your parameter file:

```
<luminosityPostprocessSet value="default recent unabsorbed recentUnabsorbed"/>
```

where one set must be specified for each luminosity specified in the `luminosityFilter` parameter. Note that set names can be reused in order to apply the same postprocessor set to multiple luminosities.

The chain of postprocessors to apply for each set is then specified as follows:

```
<stellarPopulationSpectraPostprocessRecentMethods           value="inoue2014 recent"/>
<stellarPopulationSpectraPostprocessUnabsorbedMethods       value="identity"        />
<stellarPopulationSpectraPostprocessRecentUnabsorbedMethods value="recent"          />
```

In this case we've constructed three new sets, in addition to the default set (which applies just the `inoue2014` postprocessor). The `recent` set applies both the `inoue2014` IGM absorption postprocessor, followed by the `recent` postprocessor to retain only recently emitted light. The `unabsorbed` set ignores IGM absorption entirely—it does this by using the `identity` postprocessor which leaves the spectrum unaffected. Finally, the `recentUnabsorbed` set applies only the `recent` filter while ignoring IGM absorption.

In this way it is relatively easy to extract multiple different measures of luminosity from a Galacticus model. For example, you could construct four postprocessor sets, each corresponding to one of the four different IGM absorption models (`lycSuppress`, `madau1995`, `meiksin2006`, and `inoue2014`) and apply these to the same luminosity filter to assess how luminosity depends on the IGM model used.

[6] Perhaps so that additional dust extinction can be applied to the light of recently formed stars.

6.5. Migrating Parameter Files to a New Version

The names and allowed values of parameters often change between versions of Galacticus. To permit easy and error-free migration between versions a script is provided to translate parameter files from earlier to later versions. To migrate a parameter file simply use:

```
scripts/aux/parametersMigrate.pl parameters.xml newParameters.xml
```

By default, this script will translate from the previous to the current version of Galacticus. If your parameter file contains a `version` element then this will be used to determine which version of Galacticus the parameter file was constructed for. The migration script will then migrate the parameter file through all intermediate versions to bring it into compliance with the current version. You can also specify input and output versions directly:

```
scripts/aux/parametersMigrate.pl parameters.xml newParameters.xml --inputVersion 0.9.0 --outputVersion 0
```

will convert `parameters.xml` from version 0.9.0 syntax to version 0.9.3 syntax.

6.5.1. Reionization Calculations

Galacticus can self-consistently solve for the evolution of the IGM as it becomes photoionized by light emitted by stars and AGN. To activate this calculation, include the following in your parameters file:

```
<!-- IGM evolver -->
<intergalacticMediumStateMethod  value="internal"/>
<igmPropertiesCompute            value="true"    />
<igmPropertiesTimeCountPerDecade value="10"      />

<!-- Background radiation -->
<backgroundRadiationCompute                   value="true"    />
<radiationIntergalacticBackgroundMethod       value="internal"/>
<backgroundRadiationWavelengthCountPerDecade value="50"      />
<backgroundRadiationTimeCountPerDecade        value="10"      />

<!-- Halo accretion options -->
<accretionHaloMethod value="naozBarkana2007"/>
```

The first block of parameters switches Galacticus to using an internal calculation for the state of the IGM, instructs it to solve for IGM properties as a function of time, and specifies that IGM properties should be updated 10 times per decade of cosmic time. Specifically, at each of these time intervals, solving of galaxy evolution is halted and the IGM evolved up to this time using the currently computed photoionizing background spectrum.

The second block of parameters activates an internal calculation of cosmic background radiation, in which the background is computed from the emissivities of model galaxies and AGN. The number of points at which to tabulate the background per decade of wavelength and cosmic time are specified.

Finally, the third block of parameters tells Galacticus to use the Naoz and Barkana [2007] prescription for computing gas accretion into halos from the IGM. This prescription uses the filtering mass to determine accretion rates, and will take the filtering mass from the internal IGM evolution calculation.

With these three sets of configurations, Galacticus will perform a self-consistent evolution of the IGM—in the sense that the IGM is ionized by photons emitted by model galaxies and AGN, while galaxy evolution is affected by the computed state of the model IGM. Note that, when run in this way, Galacticus needs to keep all merger trees in memory simultaneously (as they are run synchronously to allow the IGM properties to evolved alongside galaxy properties).

Once completed, data on the IGM and background radiation are written to the output file in the `igmProperties` and `backgroundRadiation` groups respectively.

7. Troubleshooting

This chapter contains guidance on solving various problems that you might encounter when running Galacticus.

7.1. Low CPU utilization with large numbers of output redshifts

If a Galacticus model is failing to make use of the majority of the available CPU cycles, and you are running a model with a large number of output redshifts the problem may be that I/O to disk is limiting the rate at which merger trees can be processed. I/O occurs through the HDF5 library which provides caching functionality. Therefore, this problem can often be mitigated by expanding the size of the HDF5 library's cache. Galacticus allows you to do this using a set of input parameters:

`[hdf5CacheElementsCount]` HDF5 limits the number of objects that it will store in its cache. Increasing this number will allow more data to be cached and potentially make disk I/O more efficient. We have had good results by setting this number to some factor (e.g. 2) times the product of the number of output redshifts and the number of properties being output in each snapshot.

`[hdf5CacheSizeBytes]` HDF5 also limits the size of the cache in bytes. We have had good results setting this to a factor of a few above the product of `[hdf5CacheElementsCount]`, the chunk size (see below) and the size of each output property (8 bytes).

`[hdf5SieveBufferSize]` HDF5 uses a sieve buffer to speed reading/writing of partial datasets. Increasing the buffer size (specified here in bytes) may improve I/O performance. We have had good results using a value of 512Kb.

`[hdf5UseLatestFormat]` Normally, HDF5 selects an internal file format to used based on maximum portability. If you set this option to `true` HDF5 will instead use the latest format that it supports—typically allowing it to employ optimizations unavailable to older versions of HDF5. Note that this can make the output file unreadable by older versions of the HDF5 library.

Additionally, you can ensure that compression is switched off in the HDF5 output by setting `[hdf5CompressionLevel]`$=-1$. Finally, adjusting the HDF5 chunk size via the `[hdf5ChunkSize]` parameter may make for more efficient I/O. HDF5 datasets are read/written in chunks of this size. Increasing the size may improve I/O performance.

8. Numerical Implementation

8.1. Timestepping Criteria

Galacticus evolves each merger tree forest by repeatedly walking the trees and evolving each node forward in time by some timestep Δt. Nodes are evolved individually such that nodes in different branches of a tree may have reached different cosmic times at any given point in the execution of Galacticus. Each node is evolved over the interval Δt using an adaptive ordinary differential equation (ODE) solver, which adjusts the smaller timesteps, δt, taken in evolving the system of ODEs to maintain a specified precision.

The choice of Δt then depends on other considerations. For example, a node should not be evolved beyond the time at which it is due to merge with another galaxy. Also, we typically don't want satellite nodes to evolve too far ahead of their host node, such that any interactions between satellite and host occur (near) synchronously.

In the remainder of this section we list all criteria used to select Δt for a node. All criteria are considered and the largest Δt consistent with all criteria is selected.

8.1.1. Tree Criteria

The following timestep criteria ensure that tree evolution occurs in a way which correctly preserves tree structure and ordering of interactions between nodes.

"Branch Segment" Criteria

For nodes which are the primary progenitor of their parent, the "branch segment" criterion asserts that

$$\Delta t \leq t_{\rm parent} - t \tag{8.1}$$

where t is current time in the node and $t_{\rm parent}$ is the time of the parent node. This ensures that primary progenitor nodes to not evolve beyond the time at which their parent (which they will replace) exists. If this criterion is the limiting criteria for Δt then the node will be promoted to replace its parent at the end of the timestep.

"Parent" Criteria

For nodes which are satellites in a hosting node the "parent" timestep criterion asserts that

$$\Delta t \leq t_{\rm host}, \tag{8.2}$$
$$\Delta t \leq \epsilon_{\rm host}(a/\dot{a}), \tag{8.3}$$

where $t_{\rm host}$ =`[timestepHostAbsolute]`, $\epsilon_{\rm host}$ =`[timestepHostRelative]`, and a is expansion factor. These criteria are intended to prevent a satellite for evolving too far ahead of the host node before the host is allowed to "catch up".

"Satellite" Criteria

For nodes which host satellite nodes, the "satellite" criterion asserts that

$$\Delta t \leq \min(t_{\rm satellite}) - t, \tag{8.4}$$

where t is the time of the host node and $t_{\rm satellite}$ are the times of all satellite nodes in the host. This criterion prevents a host from evolving ahead of any satellites.

"Sibling" Criteria

For nodes which are primary progenitors, the "sibling" criterion asserts that

$$\Delta t \leq \min(t_{\rm sibling}) - t, \tag{8.5}$$

where t is the time of the host node and $t_{\rm sibling}$ are the times of all siblings of the node. This criterion prevents a node from reaching its parent (and being promoted to replace it) before all of its siblings have reach the parent and have become satellites within it.

"Mergee" Criteria

For nodes with mergee nodes, the "mergee" criterion asserts that

$$\Delta t \leq \min(t_{\rm merge}) - t, \tag{8.6}$$

where t is the time of the host node and $t_{\rm merge}$ are the times at which the mergees will merge. This criterion prevents a node from evolving past the time at which a merger event takes place.

8.1.2. General Criteria

"Simple" Criteria

The `simple` timestep criteria assert that

$$\Delta t \leq t_{\rm simple}, \tag{8.7}$$
$$\Delta t \leq \epsilon_{\rm simple}(a/\dot{a}), \tag{8.8}$$

where $t_{\rm simple}$ =`[timestepSimpleAbsolute]`, $\epsilon_{\rm simple}$ =`[timestepSimpleRelative]`, and a is expansion factor. These criteria are intended to prevent any one node evolving over an excessively large time in one step. In general, these criteria are not necessary, as nodes should be free to evolve as far as possible unless prevented by some physical requirement. These criteria are therefore present to provide a simple example of how timestep criteria work.

"Satellite" Criteria

The `satellite` timestep criteria asserts the following for satellite nodes. If the satellite's merge target has been advanced to at least a time of $t_{\rm required} = t_{\rm satellite} + \Delta t_{\rm merge} - \delta t_{\rm merge,maximum}$ then

$$\Delta t \leq \Delta t_{\rm merge}, \tag{8.9}$$

where $t_{\rm satellite}$ is the current time for the satellite node, $\Delta t_{\rm merge}$ is the time until the satellite is due to merge and $\delta t_{\rm merge,maximum}$ is the maximum allowed time difference between merging galaxies. This ensures that the satellite is not evolved past the time at which it is due to merge. If this criterion is the limiting criteria for Δt then the merging of the satellite will be triggered at the end of the timestep.

If the merge target has not been advanced to at least $t_{\rm required}$ then instead

$$\Delta t \leq \max(\Delta t_{\rm merge} - \delta t_{\rm merge,maximum}/2, 0), \tag{8.10}$$

is asserted to ensure that the satellite does not reach the time of merging until its merge target is sufficiently close (within $\delta t_{\rm merge,maximum}$) of the time of merging.

8.1.3. Output Criteria

"History" Criteria

The `history` timestep criterion asserts that

$$\Delta t \leq t_{{\rm history},i} - t \tag{8.11}$$

where t is the current time, $t_{{\rm history},i}$ is the $i^{\rm th}$ time at which the global history (see §4.1.6) of galaxies is to be output and i is chosen to be the smallest i such that $t_{{\rm history},i} > t$. If there is no i for which $t_{{\rm history},i} > t$ this criterion is not applied. If this criterion is the limiting criterion for Δt then the properties of the galaxy will be accumulated to the global history arrays at the end of the timestep.

"Record Main Branch Evolution" Criteria

If selected, the `recordEvolution` timestep criterion asserts that

$$\Delta t \leq t_{{\rm record},i} - t \tag{8.12}$$

where t is the current time, $t_{{\rm record},i}$ is the $i^{\rm th}$ time at which the evolution of main branch galaxies is to be output and i is chosen to be the smallest i such that $t_{{\rm record},i} > t$. If there is no i for which $t_{{\rm record},i} > t$ this criterion is not applied. If this criterion is the limiting criterion for Δt then the properties of the galaxy will be recorded at the end of the timestep.

Part II.

Advanced Use

9. Constraining Galacticus

9.1. Model Accuracy

The model accuracy script processes the model described by a standard Galacticus constraints configuration file to assess how accurate the model is. Specifically, it determines the relative contribution to the covariance of each constraint arising from the finite number of merger trees run in the model and the intrinsic covariance of the observations. An accurate model should make a negligible contribution to the covariance.

To run the model accuracy script use

```
constraints/testModelAccuracy.pl <configFile>
```

where `configFile` is the name of the configuration file. The script will run the model multiple times, reducing the number of trees per decade by a factor of two each time (this is done 8 times, such that the smallest model run has a factor 128 times fewer trees than the original model). Furthermore, this sequence of models is run for three different choices for sampling halo masses: `powerLaw`, `haloMassFunction`, and `stellarMassFunction` (see §12.11.6).

For each constraint in the specified constraint compilation and accuracy measure is constructed which is the root mean squared ratio of the error arising from the finite number of trees and the intrinsic error of the constraint data.

Accuracy analysis files are written to:

```
<workDirectory>/accuracy
```

For each constraint in the compilation a plot showing accuracy measure as a function of CPU time is written to

```
<constraintLabel>_mergerTreeBuildTreesPerDecade.pdf
```

Each point in the plot is labelled with the number of merger trees per decade. An accurate model should have accuracy measure significantly below unity. This plot is also useful to see which sampling method achieves that accuracy in the least amount of CPU time.

Additionally, a report is written to:

```
<constraintLabel>Report.txt
```

This file lists, for each sampling method, the accuracy measure achieved and the CPU time taken for the largest model run.

9.2. Model Convergence

The model convergence script processes the model described by a standard Galacticus constraints configuration file to assess how well converged the model is with respect to several of Galacticus's numerical parameters. Specifically, it determines the a covariance measure for each constraint in the specified compilation file.

To run the model convergence script use

```
constraints/testConvergnce.pl <configFile>
```

where `configFile` is the name of the configuration file. The script will run the model multiple times, adjusting the value of a numerical parameter each time.

For each constraint in the specified constraint compilation a convergence measure, C, is constructed which

$$C = \sum_i \frac{(y_i - y_{\mathrm{ideal},i})^2}{\sqrt{2}\sigma^2_{\mathrm{ideal},i}} \tag{9.1}$$

where y_i is the result of the constraint, σ_i is the error on the result, and subscript "ideal" refers to the model with the most ideal value (i.e. that in the original, unmodified model) of the numerical parameter being tested (which might be the lowest value for a mass resolution, or the highest value for the maximum tree mass simulated for example). To be converged, the convergence measure should remain consistent with unity within a significant distance[1] away from the ideal value.

Convergence analysis files are written to:

```
<workDirectory>/convergence
```

For each combination of constraint in the compilation and numerical parameter a plot showing the convergence measure as a function of the numerical parameter is created in:

```
<constraintLabel>_<numericalParameter>.pdf
```

Each point in the plot has an error bar since, due to the limited number of merger trees run, the convergence measure is not known with perfect precision. A horizontal line shows the desired convergence measure of unity.

Additionally, a report is written to:

```
<constraintLabel>Report.txt
```

This file lists, for each numerical parameter, the convergence measure and its error achieved by the ideal model. Additionally a normalized measure (the measure divided by its error) is listed. This normalized measure can be approximately interpretted as the number of σ deviation from convergence.

9.3. Model Discrepancy

Model discrepancy scripts process the model described by a standard Galacticus constraints configuration file to produce an output HDF5 file which describes a particular contribution to the model discrepancy. The format of these files is

```
HDF5 "discrepancy.hdf5" {
GROUP "/" {
   DATASET "additive" {
   }
   DATASET "multiplicative" {
   }
   DATASET "covariance" {
   }
}
```

[1] "Significant distance" here requires some judgement. Typically we would like for the model results to not change significantly as the value of a numerical parameter is adjusted by at least a factor of 2 away from the ideal value.

Each of the three datasets is optional (i.e. not all need be provided for each discrepancy). The `additive` dataset gives an additive offset which will be applied to the relevant model results. The `multiplicative` dataset similarly gives a multiplicative offset which will be applied to the relevant model results. Finally, the `covariance` dataset gives the contribution from this discrepancy to the covariance matrix used in evaluating the model likelihood.

Discrepancy files are written to

```
<workDirectory>/modelDiscrepancy/<discrepancyLabel>/discrepancy<constraintLabel>.hdf5
```

Constraint scripts (see §9.5) accept a command line option `-modelDiscrepancies` which specifies the path to the `modelDiscrepancy` directory (i.e. `<workDirectory>/modelDiscrepancy`) and will search for any relevant model discrepancy files and apply them in their calculations.

9.3.1. Monte Carlo Merger Trees

Galacticus typically uses Monte Carlo-generated merger trees when being constrained to fit data. These have the advantage that they can be generated for any cosmological parameters (necessary if the cosmological parameters are to be varied as part of the constraining process) and they can be generated uniquely for each model evaluation which avoids any bias introduced by using a fixed set of halos.

However, these Monte Carlo-generated trees may not precisely capture the properties of merger trees derived from a fully non-linear calculation of gravitational collapse (e.g. as performed by an N-body simulation). Therefore it is important to assess the model discrepancy arising from this limitation.

Model discrepancy files can be generated using:

```
constraints/modelDiscrepancy/monteCarloTrees.pl config.xml
```

where `config.xml` is a standard Galacticus constraint configuration file. The script will run two sets of models, one using N-body merger trees derived from the Millennium Simulation, and a second using Monte Carlo-generated merger trees. The number of subvolumes of the Millennium Simulation to use is specified by the `subVolumeCount` option to this script (a default of 32 subvolumes is used if no number is specified). The subvolume data will be downloaded from the Millennium Simulation database if necessary.

To make a fair comparison, Millennium Simulation merger trees have their branches pruned below a mass corresponding to 20 particles, and the Monte Carlo merger trees are built with the equivalent mass resolution. Additionally, the Monte Carlo merger trees are regridded onto a set of timesteps matched to the Millennium Simulation.

A multiplicative model discrepancy is computed for each constraint included in the compilation (as specified in the configuration file) equal to the ratio of the N-body result to the Monte Carlo result. Additionally, the subvolumes of the Millennium Simulation are used to estimate the covariance in the N-body result due to the finite volume of the simulation. The result is computed for each subvolume separately and the covariance of the result between subvolumes computed. This is repeated using pairs of subvolumes, quads of subvolumes, etc. If 2^n subvolumes were used, then the covariance measured from the result combining 2^{n-3} subvolumes is used to extrapolate the covariance for all 512 subvolumes assuming that the covariance scales in inverse proportion to the number of subvolume used. Finally, the contribution of the Monte Carlo trees model to the covariance is assumed to be a diagonal matrix with elements equal to the square of the reported errors on the result of the model.

9.3.2. Fixed Virial Orbits

The orbital parameters of subhalos at the point of virial orbit crossing are usually drawn from an appropriate cosmological distribution. If instead fixed virial orbital parameters are used instead then term should be included in the model discrepancy accounting for this approximation.

Model discrepancy files can be generated using:

```
constraints/modelDiscrepancy/fixedVirialOrbits.pl config.xml
```

where `config.xml` is a standard Galacticus constraint configuration file. The script will run two models, one using fixed virial orbital parameters, and a second using variable orbital parameters using the `Benson2005` method (see §12.51.2). A multiplicative model discrepancy is computed for each constraint included in the compilation (as specified in the configuration file) equal to the ratio of the variable orbits result to the fixed orbits result. Additionally, a model discrepancy covariance is computed. This is assumed to be a diagonal matrix with elements equal to the square of the reported errors on the results of the fixed and variable orbital parameters models.

9.3.3. Jiang et al. (2008) Merger Time Scatter

The Jiang et al. [2008] algorithm for the merging times of dark matter subhalos includes drawing times from a log-normal distribution of width $\sigma = 0.4$ with median equal to their fitting function (see §12.51.1). If instead zero scatter is used then a term should be included in the model discrepancy accounting for this approximation.

Model discrepancy files can be generated using:

```
constraints/modelDiscrepancy/jiang2008MergingTimeScatter.pl config.xml
```

where `config.xml` is a standard Galacticus constraint configuration file. The script will run two models, one using the default scatter specified by the configuration file, and a second using $\sigma = 0.4$. A multiplicative model discrepancy is computed for each constraint included in the compilation (as specified in the configuration file) equal to the ratio of the $\sigma = 0.4$ and default scatter results. Additionally, a model discrepancy covariance is computed. This is assumed to be a diagonal matrix with elements equal to the sum of the square of the reported errors on the result of the default scatter and $\sigma = 0.4$ models.

9.4. Optimal Halo Mass Function Sampling

Suppose we want to fit parameters of the Galacticus model to some dataset. The basic approach is to generate large numbers of model realizations for different parameter values and see which ones best match the data. Galacticus models involve simulating individual merger trees and then adding together their galaxies to produce some overall function. The question we want to answer is, given some finite amount of computing time, what is the optimal distribution of halo masses to run when comparing to a given dataset. For example, is it better to run a volume limited sample (as one would get from an N-body simulation) or is it better to use, say, equal numbers of halos per logarithmic interval of halo mass? The following section describes how to solve this optimization problem in the specific case of fitting to the stellar mass function.

9.4.1. Li & White (2009) Stellar Mass Function

First, some definitions:

$n(M)\mathrm{d}\ln M$ is the dark matter halo mass function, i.e. the number of halos in the range M to $M + M\mathrm{d}\ln M$ per unit volume;

$\gamma(M)\mathrm{d}\ln M$ is the number of trees that we will simulate in the range M to $M + M\mathrm{d}\ln M$;

$\alpha(M_\star)$ is the error on the observed stellar mass function at mass $M_\star$;

$P(N|M_\star, M; \delta\ln M_\star)$ is the conditional stellar mass distribution function of galaxies of stellar mass $M_\star$ in a bin of width $\delta\ln M_\star$ per halo of mass M;

$t(M)$ is the CPU time it takes to simulate a tree of mass M.

To clarify, $P(N|M_\star, M; \delta \ln M_\star; \delta \ln M_\star)$ is the probability[2] to find N galaxies of mass between $M_\star$ in a bin of width $\delta \ln M_\star$ in a halo of mass M. The usual conditional stellar mass function is simply the first moment of this distribution:

$$\phi(M_\star; M)\delta \ln M_\star = \sum_{N=0}^{\infty} N P(N|M_\star, M; \delta \ln M_\star) \tag{9.2}$$

The model estimate of the stellar mass function $\Phi(M_\star)$ (defined per unit $\ln M_\star$) is

$$\Phi(M_\star) = \int_0^\infty \phi(M_\star; M) \frac{n(M)}{\gamma(M)} \gamma(M) \mathrm{d} \ln M, \tag{9.3}$$

where the $n(M)/\gamma(M)$ term is the weight assigned to each tree realization—and therefore the weight assigned to each model galaxy when summing over a model realization to construct the stellar mass function.

When computing a model likelihood, we must employ some statistic which defines how likely the model is given the data. Typically, for stellar mass functions we have an estimate of the variance in the data, $\alpha^2(M_\star)$, as a function of stellar mass (full covariance matrices are typically not provided but, ideally would be, and can be easily incorporated into this method). In that case, we can define a likelihood

$$\ln \mathcal{L} = -\frac{1}{2} \sum_i \frac{[\phi_{\mathrm{obs},i} - \phi_i]^2}{\alpha_i^2 + \sigma_i^2} \tag{9.4}$$

where the sum is taken over all data points, i, and σ_i^2 is the variance in the model estimate and is given by

$$\sigma^2(M_\star) = \langle [\phi(M_\star) - \bar{\phi}(M_\star)]^2 \rangle, \tag{9.5}$$

where $\phi(M_\star)$ is the realization from a single model and $\bar{\phi}(M_\star)$ is the model expectation from an infinite number of merger tree realizations and the average is taken over all possible model realizations. Since the contributions from each merger tree are independent,

$$\sigma^2(M_\star) = \sum_i \zeta_i^2(M_\star; M) \tag{9.6}$$

where $\zeta_i^2(M_\star; M)$ is the variance in the contribution to the stellar mass function from tree i. This in turn is given by

$$\zeta^2(M_\star; M) = \psi^2(M_\star; M) \left[\frac{n(M)}{\gamma(M)}\right]^2, \tag{9.7}$$

where $\psi^2(M_\star; M)$ is the variance in the conditional stellar mass function. In the continuum limit this becomes

$$\sigma^2(M_\star) = \int_0^\infty \psi^2(M_\star; M) \left[\frac{n(M)}{\gamma(M)}\right]^2 \gamma(M) \mathrm{d} \ln M. \tag{9.8}$$

Model variance artificially increases the likelihood of a given model. We would therefore like to minimize the increase in the likelihood due to the model variance:

$$\Delta 2 \ln \mathcal{L} = \sum_i \frac{[\phi_{\mathrm{obs},i} - \phi_i]^2}{\alpha_i^2} - \frac{[\phi_{\mathrm{obs},i} - \phi_i]^2}{\alpha_i^2 + \sigma_i^2} \tag{9.9}$$

[2]To put it another way, $P(N|M_\star, M; \delta \ln M_\star)$ is closely related to the commonly used Halo Occupation Distribution.

Of course, we don't know the model prediction, ϕ_i, in advance[3]. However, if we assume that a model exists which is a good fit to the data then we would expect that $[\phi_{\mathrm{obs},i} - \phi_i]^2 \approx \alpha_i^2$ on average. In that case, the increase in likelihood due to the model is minimized by minimizing the function[4]

$$F[\gamma(M)] = \sum_i \frac{\alpha_i^2}{\alpha_i^2 + \sigma_i^2}. \tag{9.10}$$

If the bins all have the same $\delta \ln M_\star$ we can turn the sum into an integral

$$F[\gamma(M)] = \int_0^\infty \frac{\alpha(M_\star)^2}{\alpha(M_\star)^2 + \sigma(M_\star)^2} \mathrm{d} \ln M_\star. \tag{9.11}$$

Obviously, the answer is to make $\gamma(M) = \infty$, in which case $F[\gamma(M)] = 0$. However, we have finite computing resources. The total time to run our calculation is

$$\tau = \int_0^\infty t(M)\gamma(M) \mathrm{d} \ln M. \tag{9.12}$$

We therefore want to minimize $F[\gamma(M)]$ while keeping τ equal to some finite value. We can do this using a Lagrange multiplier and minimizing the function

$$F[\gamma(M)] = \int_0^\infty \frac{\alpha(M_\star)^2}{\alpha(M_\star)^2 + \sigma(M_\star)^2} \mathrm{d} \ln M_\star + \int_0^\infty \lambda\gamma(M)t(M) \mathrm{d} \ln M. \tag{9.13}$$

Finding the functional derivative and setting it equal to zero gives:

$$\gamma(M) = \sqrt{\frac{\xi(M)}{\lambda t(M)}}, \tag{9.14}$$

in the limit where[5] $\sigma(M_\star) \ll \alpha(M_\star)$, and where

$$\xi(M) = n^2(M) \int_{-\infty}^\infty \frac{\psi^2(M_\star; M)}{\alpha^2(M_\star)} \mathrm{d} \ln M_\star. \tag{9.15}$$

The values of λ and $\delta \ln M_\star$, and the normalization of $t(M)$ are unimportant here since we merely want to find the optimal shape of the $\gamma(M)$ function—we can then scale it up or down to use the available time.

Figure 9.2 shows the function $\gamma(M)$ obtained by adopting a model conditional stellar mass function which is a sum of central and satellite terms. Specifically, we use the model of Leauthaud et al. [2011] which is constrained to match observations from the COSMOS survey. In their model[6]:

$$\langle N_\mathrm{c}(M_\star|M) \rangle \equiv \int_{M_\star}^\infty \phi_\mathrm{c}(M'_\star) \mathrm{d} \ln M'_\star = \frac{1}{2} \left[1 - \mathrm{erf} \left(\frac{\log_{10} M_\star - \log_{10} f_\mathrm{SHMR}(M)}{\sqrt{2} \sigma_{\log M_\star}} \right) \right]. \tag{9.16}$$

Here, the function $f_\mathrm{SHMR}(M)$ is the solution of

$$\log_{10} M = \log_{10} M_1 + \beta \log_{10} \left(\frac{M_\star}{M_{\star,0}} \right) + \frac{(M_\star/M_{\star,0})^\delta}{1 + (M_\star/M_{\star,0})^{-\gamma}} - 1/2. \tag{9.17}$$

[3] Below, we will adopt a simple empirical model for $\phi(M_\star)$. However, it should not be used here since we will in actuality be computing the likelihood from the model itself.

[4] This can be seen intuitively: we are simply requring that the variance in the model prediction is small compared the the variance in the data.

[5] This is the limit in which we would like our results to be.

[6] This integral form of the conditional stellar mass function is convenient here since it allows for easy calcualtion of the number of galaxies expected in the finite-width bins of the observed stellar mass function.

Table 9.1.: Parameters of the conditional stellar mass function fit.

Parameter	Value
$\alpha_{\rm sat}$	1.0
$\log_{10} M_1$	12.120
$\log_{10} M_{\star,0}$	10.516
β	0.430
δ	0.5666
γ	1.53
$\sigma_{\log M_\star}$	0.206
$B_{\rm cut}$	0.744
$B_{\rm sat}$	8.00
$\beta_{\rm cut}$	−0.13
$\beta_{\rm sat}$	0.859

For satellites,

$$\langle N_{\rm s}(M_\star|M)\rangle \equiv \int_{M_\star}^{\infty} \phi_{\rm s}(M'_\star){\rm d}\ln M'_\star = \langle N_{\rm c}(M_\star|M)\rangle \left(\frac{f^{-1}_{\rm SHMR}(M_\star)}{M_{\rm sat}}\right)^{\alpha_{\rm sat}} \exp\left(-\frac{M_{\rm cut}}{f^{-1}_{\rm SHMR}(M_\star)}\right), \quad (9.18)$$

where

$$\frac{M_{\rm sat}}{10^{12}M_\odot} = B_{\rm sat}\left(\frac{f^{-1}_{\rm SHMR}(M_\star)}{10^{12}M_\odot}\right)^{\beta_{\rm sat}}, \quad (9.19)$$

and

$$\frac{M_{\rm cut}}{10^{12}M_\odot} = B_{\rm cut}\left(\frac{f^{-1}_{\rm SHMR}(M_\star)}{10^{12}M_\odot}\right)^{\beta_{\rm cut}}. \quad (9.20)$$

We use the best fit parameters from the `SIG_MOD1` method of Leauthaud et al. [2011] for their z_1 sample, but apply a shift of -0.2 dex in masses to bring the fit into line with the $z = 0.07$ mass function of Li and White [2009]. The resulting parameter values are shown in Table 9.1.

We assume that $P_{\rm s}(N|M_\star, M; \delta\ln M_\star)$ is a Poisson distribution while $P_{\rm c}(N|M_\star, M; \delta\ln M_\star)$ has a Bernoulli distribution, with each distribution's free parameter fixed by the constraint of eqn. (9.2), and the assumed forms for $\phi_{\rm c}$ and $\phi_{\rm s}$.

The errors in the Li and White [2009] observed stellar mass function are well fit by (see Fig. 9.1):

$$\alpha(M_\star) = 10^{-3}\left(\frac{M_\star}{4.5\times10^{10}M_\odot}\right)^{-0.3}\exp\left(-\frac{M_\star}{4.5\times10^{10}M_\odot}\right) + 10^{-7}, \quad (9.21)$$

and the tree processing time in Galacticus can be described by:

$$\log_{10} t(M) = \sum_{i=0}^{2} C_i[\log_{10} M]^i \quad (9.22)$$

with $C_0 = -0.73$, $C_1 = -0.20$ and $C_2 = 0.035$.

The resulting optimal sampling density curve is shown in Fig. 9.2 and is compared to weighting by the halo mass function (i.e. the result of sampling halos at random from a representative volume). Optimal sampling gives less weight to low mass halos (since a sufficient accuracy can be obtained without the need to run many tens of thousands of such halos) and to high mass halos which are computationally expensive.

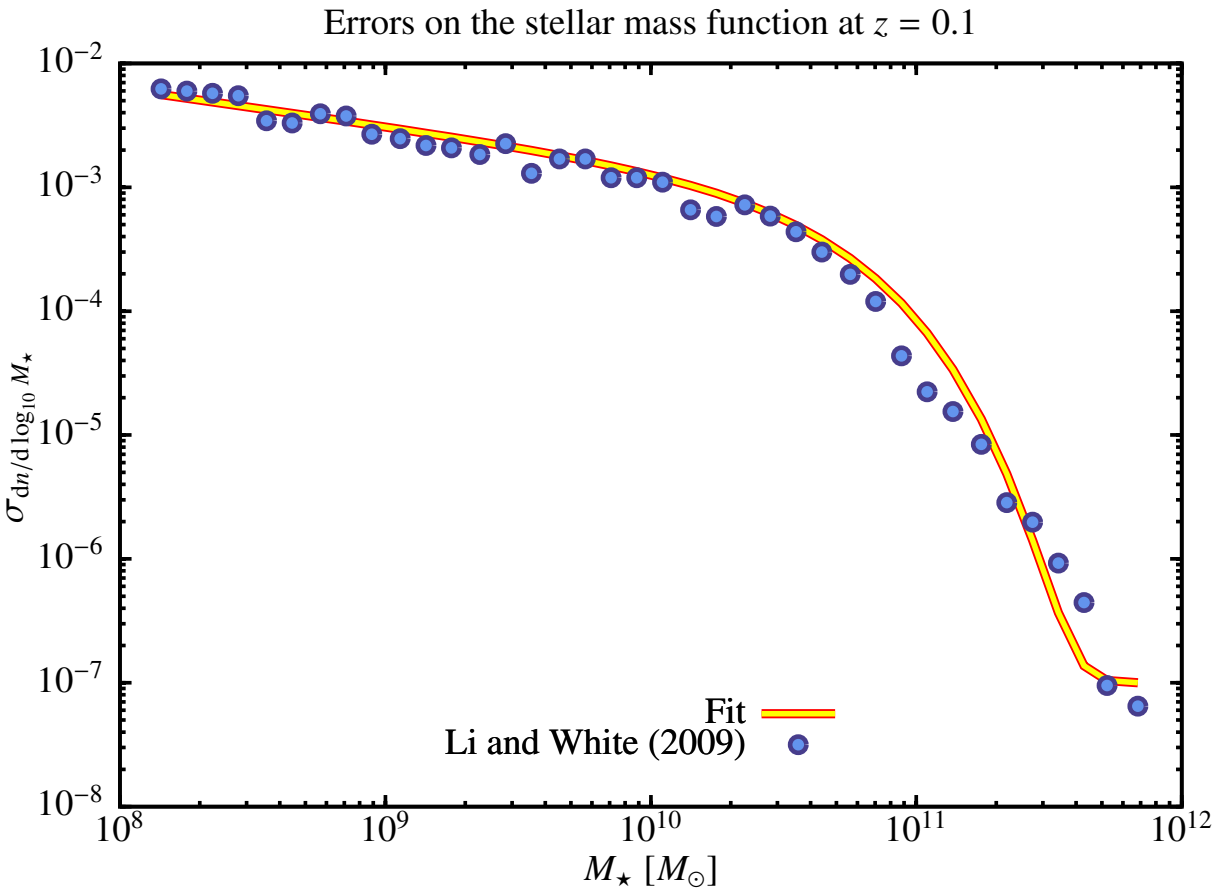

Figure 9.1.: Errors on the Li and White [2009] stellar mass funtion (points) and the fitting function (line) given by eqn. (9.21).

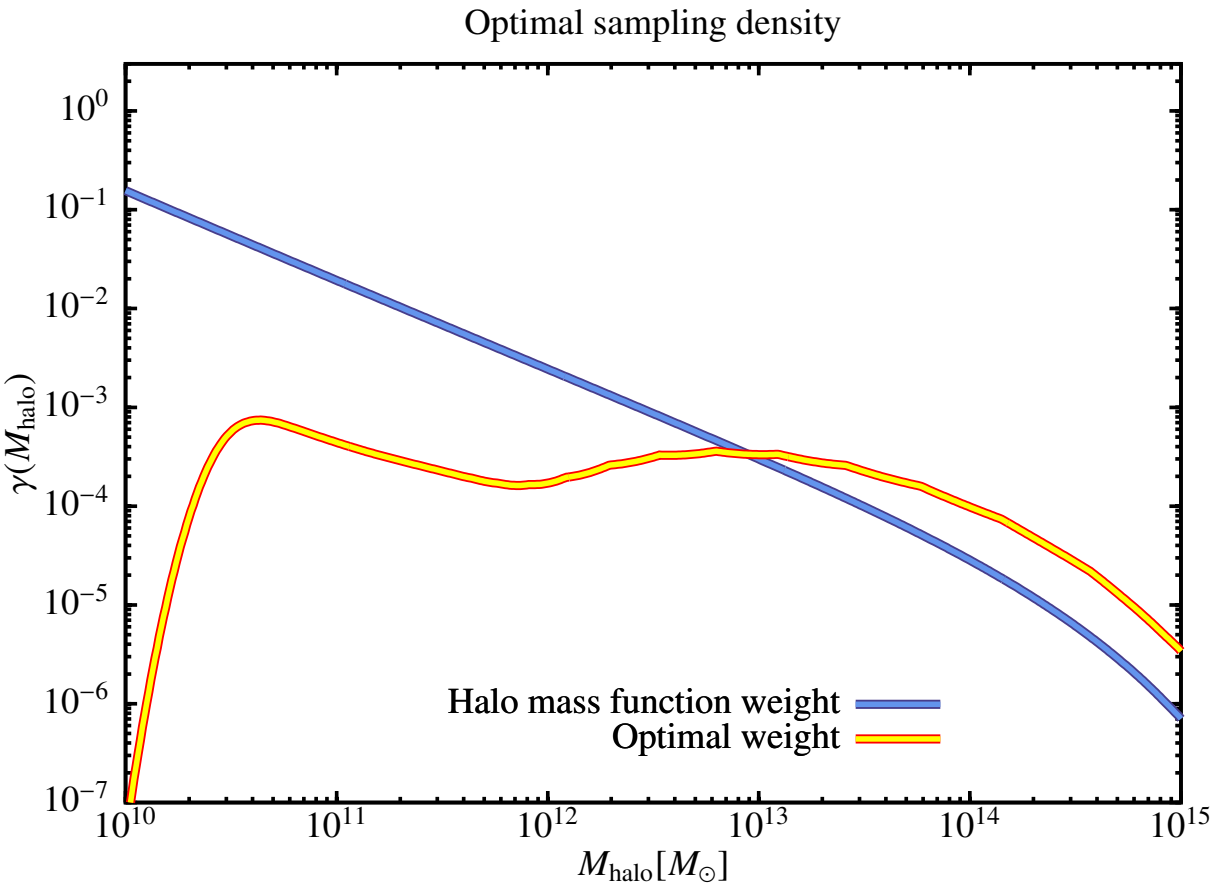

Figure 9.2.: Optimal weighting (yellow line) compared with weighting by the dark matter halo mass function (i.e. sampling halos at random from a representative volume; blue line). Sampling densities have been normalized to unit compute time.

9.4.2. Refining by Other Merger Tree Statistics

Since building merger trees is relatively fast, while solving the baryonic physics is slow it may be advantageous to non-uniformly sample the distribution of merger trees at fixed merger tree mass, M. For example, we could assign some measure of formation history to each merger tree, such as the time since the last major merger, τ. The halo mass function then becomes $n(M,\tau)$ (which can be computed by simulating large numbers of trees), and the tree sampling function becomes $\gamma(M,\tau)$. We'd then need to know the stellar mass function conditioned on both M and τ, $\phi_\star(M_\star|M,\tau)$. Given these, the above approach could be easily generalized to determine an optimal $\gamma(M,\tau)$. Then, after generating a merger tree, we'd first compute τ. If a sufficient number of trees in that τ interval had already been computed, then we'd simply drop that tree and compute another one. The speed up here would depend on how fast building trees is relative to solving baryonic physics and what fraction of trees you discard. In principle, the trees could be generated, sampled and stored in advance so that we'd already have an optimally distributed set of trees in M and τ that could be used for each model run.

9.5. Constraints

Any constraint which can be applied to Galacticus is defined by two files, a configuration file and a likelihood script, which must be placed in `constraints/constraints` and `constraints/scripts` respectively.

9.5.1. Configuration File

The configuration file should have the form:

```
<!-- Defines a constraint to match some data. -->
<constraint>
  <name>Long-form name of this constraint</name>
  <label>shortLabelForThisConstraint</label>
  <outputRedshift>0.07</outputRedshift>
  <outputRedshift>1.00</outputRedshift>
  <haloMassResolution>5.0e9</haloMassResolution>
  <haloMassMinimum>2.0e10</haloMassMinimum>
  <haloMassMaximum>2.0e14</haloMassMaximum>
  <analysis>constraints/scripts/myAnalysisScript.pl</analysis>
  <luminosity>
    <filter>UKIRT_K</filter>
    <redshift>0.0</redshift>
    <frame>rest</frame>
  </luminosity>
  <luminosity>
    <filter>UKIRT_K</filter>
    <redshift>1.0</redshift>
    <frame>observed</frame>
  </luminosity>
  <optionOn>outputMainBranchStatus</optionOn>
  <optionOn>outputDensityContrastData</optionOn>
  <parameter>
   <name>outputDensityContrastValues</name>
   <value>200.0</value>
```

```
   <accumulation>unique</accumulation>
  </parameter>
</constraint>
```

The `name` and `label` are used to describe the constraint (`label` is used as a suffix in file names so should not contain spaces or other characters which might cause problems in file names).

The remaining elements describe the requirements for this constraint. `haloMassResolution` specifies the maximum resolution in mergers trees that still allows this constraint to be computed accurately. Similarly, `haloMassMinimum` and `haloMassMaximum` specify the required range of halo masses to simulate to allow this constraint to be computed accurately.

One or more `outputRedshift` elements may be present, each specifying a redshift at which output is required for this constraint. Similarly, one or more `luminosity` elements may be present, each of which specifies a luminosity which must be computed for this constraint. Each `luminosity` must contain a specification of `filter`, `redshift`, and `frame` to define which luminosity is to be computed.

One or more `optionOn` elements may be present. Each element must specify the name of a Galacticus input parameter. That parameter will be set to `true` in the Galacticus input parameter file.

Finally, arbitrary other parameter may be set using the standard `parameter` element which should give the `name` and `value` for the parameter. Optionally, an `accumulation` element may also be specified for each `parameter`. This controls how values of the parameter are to be accumulated if set by more than one constraint. An accumulation of `overwrite` will simply overwrite any previously set values. An accumulation of `combine` will concatenate all values set by different constraints. Finally, an accumulation of `unique` will concatenate all values set by different constraints and then filter out any duplicates.

When multiple constraints are used, their requirements are automatically combined.

9.5.2. Likelihood Script

The likelihood script for a constraint is required to perform several tasks, controlled by command line options. The script should accept the following command line syntax:

```
myScript.pl <galacticusFile> [options...]
```

where `galacticusFile` is the file name of the Galacticus model for which the likelihood calculation should be performed. The following options must be supported by the script:

`-plotFile <fileName>` If this option is present, the script should generate a plot showing the constraint and the model result and write it to `fileName`.

`-outputFile <fileName>` If this option is present, the script should compute the log-likelihood of the model given the constraint and write it to `fileName` using the format

```
<constraint>
 <logLikelihood>-123</logLikelihood>
</constraint>
```

`-accuracyFile <fileName>` If this option is present, the script should write an XML file giving details of the accuracy of the model results relative to the observational errors using the format

```
<accuracy>
 <x>...</x>
 .
 .
 .
```

```
 <x>...</x>
 <yModel>...</yModel>
 .
 .
 .
 <yModel>...</yModel>
 <yData>...</yData>
 .
 .
 .
 <yData>...</yData>
 <errorModel>...</errorModel>
 .
 .
 .
 <errorModel>...</errorModel>
 <errorData>...</errorData>
 .
 .
 .
 <errorData>...</errorData>
</accuracy>
```

In this file the `yModel` and `yData` elements should give the values of the model result and the comparable data respectively, while `errorModel` and `errorData` should give an estimate of the errors on these quantities. In the case of the model error this should include only the contribution arising from the finite number of merger trees simulated. This file will be used to judge whether the model is running sufficient merger trees such that the likelihood is not dominated by these errors. The `x` elements are optional but can be used to give the parameter values associated with each model result.

`-resultFile <fileName>` If this option is present, the script should write an XML file giving details of the result of the model using the format

```
<accuracy>
 <x>...</x>
 .
 .
 .
 <x>...</x>
 <y>...</y>
 .
 .
 .
 <y>...</y>
 <error>...</error>
 .
 .
 .
 <error>...</error>
</accuracy>
```

In this file the `y` elements should give the values of the model result, while the `error` elements should give an estimate of the errors on these results. The error should include only the contribution arising from the finite number of merger trees simulated. This file will be used to judge whether the model result is converged with respect to various numerical parameters in Galacticus. The `x` elements are optional but can be used to give the parameter values associated with each model result.

`-modelDiscrepancies <path>` If this option is present, the script should scan `path`. For each directory found in `path` the script should check for the existance of a file named `discrepancy<label>.hdf5` where `label` is the label given for this constraint in its configuration file (see §9.5.1). If present, the model discrepancy given in that file should be applied to the likelihood calculation. See §9.3 for a description of the structure of the discrepancy files.

9.5.3. Available Constraints

Methodology for Mass Functions

In general, for constraints corresponding to mass functions (whether stellar mass or HI mass), the covariance matrix of the observational data is determined using the analytic model of Smith [2012]. This requires knowledge of both the survey geometry (angular mask and radial extent as a function of mass) and of the halo occupation distribution (HOD) of the observed galaxies.

Details of the survey geometry and depth are given for each individual constraints. Computing the large-scale structure contribution to the covariance function requires integration of the non-linear matter power spectrum over the Fourier transform of the survey window function. We use the method of Peacock and Dodds [1996] to determine the non-linear matter power spectrum, because of its simplicity and speed. We have checked that using a more accurate non-linear matter power spectrum (e.g. Lawrence et al. 2010) makes negligible difference to our results. If the angular power spectrum of the survey mask is available[7], this is used to compute the relation

$$\sigma^2(M_\mu, M_\nu) = \frac{2}{\pi V_\mu V_\nu} \int_0^\infty \mathrm{d}k\, k^{-4} P(k) \sum_i \sum_j \sum_{\ell=0}^\infty (2\ell+1) C_\ell^{ij} R_\ell^i(kr_{\mu 0}, kr_{\mu 1}) R_\ell^j(kr_{\nu 0}, kr_{\nu 1}), \tag{9.23}$$

where $(2\ell+1)C_\ell^{ij} = \sum_{m=-\ell}^{+\ell} \Psi_{\ell m}^i \Psi_{\ell m}^{j*}$, $\Psi_{\ell m}^i$ are the spherical harmonic coefficients of the i^{th} field of the survey, V is the maximum distance to which a galaxy of mass M can be seen, $P(k)$ is the nonlinear power spectrum and

$$R_\ell(x_0, x_1) \equiv \int_{x_0}^{x_1} x^2 j_\ell(x) \mathrm{d}x = \sqrt{\pi} 2^{-2-\ell} \Gamma\left(\frac{1}{2}[3+\ell]\right) \left[x^{3+\ell}\, {}_1\widetilde{F}_2\left(\frac{1}{2}[3+\ell]; \ell+\frac{3}{2}, \frac{1}{2}(5+\ell); -\frac{x^2}{4}\right)\right]_{x_0}^{x_1}, \tag{9.24}$$

where ${}_1\widetilde{F}_2$ is the regularized generalized hypergeometric function. In other cases, where the angular power spectrum is not available, the survey geometry is realized on a grid which is when Fourier transformed to obtain the appropriate window function.

To find a suitable HOD to describe the observed galaxies we adopt the model of Behroozi et al. [2010]. This is an 11 parameter model which describes separately the numbers of satellite and central galaxies occupying a halo of given mass—the reader is referred to Behroozi et al. [2010] for a complete description of the functional form of this parametric HOD. An Markov Chain Monte Carlo (MCMC) approach is used to to constrain the HOD parameters to fit the observational data. We use a likelihood

$$\ln \mathcal{L} = -\frac{1}{2} \Delta \cdot \mathcal{C}^{-1} \cdot \Delta^{\mathrm{T}} - \frac{N}{2} \ln(2\pi) - \frac{\ln |\mathcal{C}|}{2}, \tag{9.25}$$

[7] Typically if the survey geometry is defined by mangle polygons, allowing the angular power spectrum to be found using the mangle `harmonize` utility.

where N is the number of bins in the mass function, $\mathcal{C}$ is the covariance matrix of the observed mass function, and $\Delta_i = \phi_i^{(\mathrm{HOD})} - \phi_i^{(\mathrm{observed})}$. Of course, it is precisely this covariance matrix, $\mathcal{C}$, that we are trying to compute. We therefore adopt an iterative approach as follows:

1. make an initial estimate of the covariance matrix, assuming that only Poisson errors contribute (the covariance matrix is therefore diagonal, and the terms are easily computed from the measured mass function and the survey volume as a function of stellar mass);
2. find the maximum likelihood parameters of the HOD given the observed mass function and the current estimate of the covariance matrix;
3. using this HOD and the framework of Smith [2012], compute a new estimate of the covariance matrix, including all three contributions;
4. repeat steps 2 and 3 until convergence in the covariance matrix is achieved.

In practice we find that this procedure often leads to an HOD and covariance matrix which oscillate between two states in successive iterations. The differences in the covariance matrix are relatively small however, so we choose to conservatively adopt the covariance matrix with the larger values.

Methodology for Correlation Functions

For constraints corresponding to (possibly, projected) correlation functions, the model expectation is computed using the halo model Cooray and Sheth [2002]. For each model halo, each galaxy (satellite and central) is assessed to see if it meets the criteria for inclusion in the sample. Where the sample includes mass limits (either just a lower limit, or lower and upper limits) each galaxy is assigned a probability of inclusion in the sample based on its mass and the random error in mass. Thus, each halo is characterized by the probability of having a central galaxy in the sample, $p^{(\mathrm{c})}$, and N probabilities, $p_i^{(\mathrm{s})}$, of each satellite galaxy being in the sample. We assume binomial statistics for each galaxy's probability of inclusion, and further assume that these probabilities are uncorrelated. Therefore, the contribution of the halo to the one- and two-halo terms of the power spectrum in the halo model are:

$$\Delta P^{1\mathrm{h}}(k) = \frac{w}{n_{\mathrm{gal}}^2}\left[p^{(\mathrm{c})}\sum_{i=1}^{N} p^{(\mathrm{s})} u(k|M) + \sum_{k=0}^{N} k(k-1) P\left(p_i^{(\mathrm{s})},\ldots,p_N^{(\mathrm{s})}\right) u(k|M)^2\right] \tag{9.26}$$

and

$$\Delta\sqrt{P^{2\mathrm{h}}}(k) = \frac{w}{n_{\mathrm{gal}}}\sqrt{P^{\mathrm{lin}}}(k) b(M) u(k|M)\left[p^{(\mathrm{c})} + \sum_{i=1}^{N} p_i^{(\mathrm{s})}\right], \tag{9.27}$$

respectively, where w is the weight of the halo (i.e. the number of such model halos expected per unit volume), $b(M)$ is the bias of halos of mass M, $u(k|M)$ is the Fourier-transform of the halo density profile, and $P_{\mathrm{lin}}(k)$ is the linear theory power spectrum, and $P(p_1,\ldots,p_N)$ is the Poisson binomial distribution for N events with probabilities $p_1,\ldots,p_N$. The contribution of the halo to the galaxy density, n_{gal}, is simply $\Delta n_{\mathrm{gal}} = w\left[p^{(\mathrm{c})} + \sum_{i=1}^{N} p_i^{(\mathrm{s})}\right]$.

Li & White (2009) SDSS Stellar Mass Function

This constraint utilizes the stellar mass function for $z \approx 0.07$ galaxies measured by Li and White [2009] from the Sloan Digitial Sky Survey (SDSS). The mass function reported by Li and White [2009] is converted to the appropriate Hubble constant for the given Galacticus model (assuming that masses scale as H_0^{-2} and volumes as H_0^3)—no adjustment is made for cosmological parameters given the low redshift of the sample.

Given a Galacticus model, total stellar masses of model galaxies are adjusted using:

$$M_\star \rightarrow \mathbf{GS} M_\star \tag{9.28}$$

where the $\mathbf{S}$ operator is a multiplicative factor accounting for systematic errors in stellar mass determination and is equal to [Behroozi et al., 2010]

$$\log_{10} S = \mu + \kappa \log_{10} \left(\frac{M_\star}{10^{11.3} M_\odot} \right) \tag{9.29}$$

where $\mu =$ `[sdssStellarMassFunctionZ0.07StellarMassSystematicMu]`, $\kappa =$ `[sdssStellarMassFunctionZ0.07StellarMa` and the $\mathbf{G}$ operator is a multiplicative factor drawn from a log-normal distribution of width 0.07 dex for each galaxy to mimic the effects of random errors on stellar masses (motivated by the discussion of Behroozi et al. [2010]).

The model masses are then used to construct a mass function by binning into a histogram using the masses reported by Li and White [2009] (modified as described above) as the centers of the bins (with bin boundaries placed at the geometric means of consecutive bin centers).

If the `-modelDiscrepancies` option is given, then any multiplicative or additive discrepancies found are applied to the model mass function, and any additional covariance is added to the covariance matrix.

The covariance matrix is computed as

$$\mathbf{C} = \mathbf{C}_\mathrm{obs} + \mathbf{C}_\mathrm{model,random} + \sum_i \mathbf{C}_\mathrm{discrepancy},_i, \tag{9.30}$$

where $\mathbf{C}_\mathrm{obs}$ is the covariance matrix of the observational data, $\mathbf{C}_\mathrm{model,random}$ is the covariance matrix of the model arising from random noise (due to the finite number of trees simulated—see §9.5.3 for a description of how this covariance matrix is estimated), and $\mathbf{C}_\mathrm{discrepancy},_i$ is the covariance due to the i^th model discrepancy.

The model likelihood is then computed using:

$$\mathcal{L} = \frac{1}{\sqrt{(2\pi)^n |\mathbf{C}|}} \exp \left[-\frac{1}{2} \Delta \mathbf{C}^{-1} \Delta \right], \tag{9.31}$$

where $\Delta_i = \Phi_\mathrm{model},_i - \Phi_\mathrm{observed},_i$ is the difference between the model and observed mass functions, and n is the number of points in the mass function histogram.

Caputi et al. (2011) UKIDSS UDS Stellar Mass Function

This constraint utilizes the stellar mass functions for $z = 3$ to 5 galaxies measured by Caputi et al. [2011] from the UKISS UDS survey. If the `-modelDiscrepancies` option is given, then any multiplicative or additive discrepancies found are applied to the model mass function, and any additional covariance is added to the covariance matrix.

Given a Galacticus model, total stellar masses of model galaxies are adjusted using:

$$M_\star \rightarrow \mathbf{GS} M_\star \tag{9.32}$$

where the $\mathbf{S}$ operator is a multiplicative factor accounting for systematic errors in stellar mass determination and is equal to [Behroozi et al., 2010]

$$\log_{10} S = \mu + \kappa \log_{10} \left(\frac{M_\star}{10^{11.3} M_\odot} \right) \tag{9.33}$$

where $\mu =$ `[ukidssUdsStellarMassFunctionZX.XXXStellarMassSystematicMu]`, $\kappa =$ `[ukidssUdsStellarMassFunctionZX.` and the $\mathbf{G}$ operator is a multiplicative factor drawn from a log-normal distribution of width 0.173 dex

for each galaxy to mimic the effects of random errors on stellar masses (which accounts for errors in photometric redshifts, $\delta z/(1+z)$ of $\sigma = 0.05$ as reported by Caputi et al. [2011] along with an additional 0.087 dex added in quadrature to account for SED fitting errors, judged from Figure 8 of Caputi et al. [2011]).

The covariance matrix is computed as

$$\mathbf{C} = \mathbf{C}_{\rm obs} + \mathbf{C}_{\rm model,random} + \sum_i \mathbf{C}_{{\rm discrepancy},i}, \tag{9.34}$$

where $\mathbf{C}_{\rm obs}$ is the covariance matrix of the observational data, $\mathbf{C}_{\rm model,random}$ is the covariance matrix of the model arising from random noise (due to the finite number of trees simulated, and $\mathbf{C}_{{\rm discrepancy},i}$ is the covariance due to the $i^{\rm th}$ model discrepancy.

The model likelihood is then computed using:

$$\mathcal{L} = \frac{1}{\sqrt{(2\pi)^n |\mathbf{C}|}} \exp\left[-\frac{1}{2}\Delta \mathbf{C}^{-1} \Delta\right], \tag{9.35}$$

where $\Delta_i = \Phi_{{\rm model},i} - \Phi_{{\rm observed},i}$ is the difference between the model and observed mass functions, and n is the number of points in the mass function histogram.

The survey window function is determined from the set of galaxy positions provided by Caputi (private communication), by finding a suitable bounding box and then cutting out empty regions (corresponding to regions that were removed around bright stars). A set of random points are then found within this mask and are used to find the Fourier transform of the survey volume.

To estimate the depth of the Caputi et al. [2011] sample as a function of galaxy stellar mass we make use of semi-analytic models in the Millennium Database. Specifically, we use the semi-analytic models (SAMs) of Guo et al. [2011] and Henriques et al. [2012] specifically the `Guo2010a..MR` and `Henriques2012a.wmap1.BC03_001` tables in the Millennium Database. For each snapshot in the database, we extract the stellar masses and observed-frame IRAC 4.5μm apparent magnitudes (including dust extinction), and determine the median apparent magnitude as a function of stellar mass. Using the limiting apparent magnitude of the Caputi et al. [2011] sample, $i_{4.5} = 24$, we infer the corresponding absolute magnitude at each redshift and, using our derived apparent magnitude–stellar mass relation, infer the corresponding stellar mass.

The end result of this procedure is the limiting stellar mass as a function of redshift, accounting for k-corrections, evolution, and the effects of dust. Figure 9.5.3 shows the resulting relation between stellar mass and the maximum redshift at which such a galaxy would be included in the sample. Points indicate measurements from the SAM, while the line shows a polynomial fit:

$$z(M_\star) = -56.247 + 5.881 m, \tag{9.36}$$

where $m = \log_{10}(M_\star/M_\odot)$. We use this polynomial fit to determine the depth of the sample as a function of stellar mass.

Finally, the incompleteness of the observational sample (which is required when estimating the Poisson contribution to the covariance matrix) is found from the 50% and 80% completeness masses, M_{50} and M_{80} respectively, given in Fig. 4 of Caputi et al. [2011]. Specifically, we assume that, at a given mass M, the number of photons being received from a galaxy can be modeled as a Gaussian distribution with mean fM and variance $fM + \mu$, where μ is the number of photons ariving from the sky. The fraction of sources of mass M that will be detected at more than $n\sigma$ above the background is then

$$\begin{aligned} f(M) &= \int_{n\sqrt{mu}}^{\infty} \frac{1}{\sqrt{2\pi}\sqrt{fM+\mu}} \exp\left(-\frac{[S-fM]^2}{2[fM+\mu]}\right) {\rm d}S \\ &= \frac{1}{2}\left[1 - {\rm erf}\left(\frac{x(M)}{\sqrt{2}}\right)\right], \end{aligned} \tag{9.37}$$

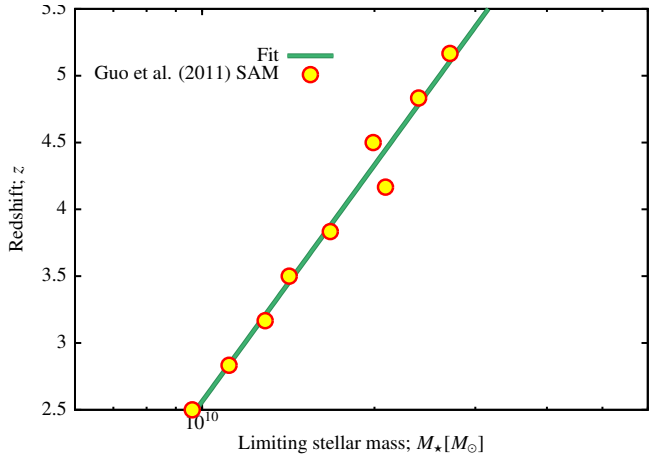

Figure 9.3.: The maximum redshift at which a galaxy of given stellar mass can be detected in the sample of Caputi et al. [2011]. Points show the results obtained using the Henriques et al. [2012] model from the Millennium Database, while the lines shows a polynomial fit to these results (given in eqn. 9.36).

where $x(M) = (n\sqrt{mu} - fM)/(\mu + fM)^{1/2}$. Given $f(M_{50}) = 0.5$ and $f(M_{80}) = 0.8$ we can solve for f and μ, and then compute the completeness in each mass using eqn. (9.37). The resulting completeness curves are shown in Fig. 9.5.3. Note that the model of eqn. (9.37) is clearly an oversimplification, but should capture the expected behavior of the completeness and, since it is fit to the 50% and 80% completenesses reported by Caputi et al. [2011]—which were computed using detailed simulations—should work sufficiently well.

Bernardi et al. (2013) SDSS Stellar Mass Function

This constraint utilizes the stellar mass function for $z \approx 0.07$ galaxies measured by Bernardi et al. [2013] from the SDSS survey. The mass function, and its covariance matrix $\mathbf{C}_{\rm model}$, is calculated as described in §4.4.1, typically on-the-fly as GALACTICUS is run. If the `-modelDiscrepancies` option is given, then any multiplicative or additive discrepancies found are applied to the model mass function, and any additional covariance is added to the covariance matrix.

The covariance matrix is computed as

$$\mathbf{C} = \mathbf{C}_{\rm obs} + \mathbf{C}_{\rm model,random} + \sum_i \mathbf{C}_{{\rm discrepancy},i}, \tag{9.38}$$

where $\mathbf{C}_{\rm obs}$ is the covariance matrix of the observational data, $\mathbf{C}_{\rm model,random}$ is the covariance matrix of the model arising from random noise (due to the finite number of trees simulated, and $\mathbf{C}_{{\rm discrepancy},i}$ is the covariance due to the $i^{\rm th}$ model discrepancy.

The model likelihood is then computed using:

$$\mathcal{L} = \frac{1}{\sqrt{(2\pi)^n |\mathbf{C}|}} \exp\left[-\frac{1}{2}\Delta \mathbf{C}^{-1} \Delta\right], \tag{9.39}$$

where $\Delta_i = \Phi_{{\rm model},i} - \Phi_{{\rm observed},i}$ is the difference between the model and observed mass functions, and n is the number of points in the mass function histogram.

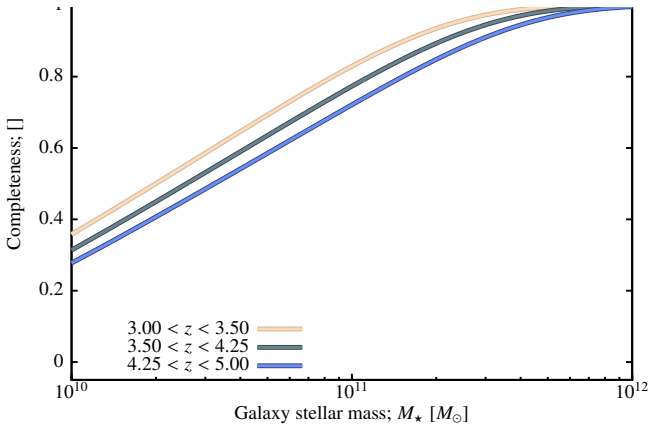

Figure 9.4.: The completeness as a function of stellar mass in the survey of Caputi et al. [2011]. Curves are computed using eqn. (9.37) with parameters fit to the reported 50% and 80% completeness masses from Caputi et al. [2011].

When computing the Poisson term in the covariance matrix, the number of galaxies per bin is determined directly from the mass function, assuming a 91% completeness[8]. The survey geometry and depth is described in §12.59.2.

Baldry et al. (2012) GAMA Stellar Mass Function

This constraint utilizes the stellar mass function for $z < 0.06$ galaxies measured by Baldry et al. [2012] from the GAMA survey. The mass function, and its covariance matrix $\mathbf{C}_{\rm model}$, is calculated as described in §4.4.1, typically on-the-fly as GALACTICUS is run. If the `-modelDiscrepancies` option is given, then any multiplicative or additive discrepancies found are applied to the model mass function, and any additional covariance is added to the covariance matrix.

The covariance matrix is computed as

$$\mathbf{C} = \mathbf{C}_{\rm obs} + \mathbf{C}_{\rm model,random} + \sum_i \mathbf{C}_{{\rm discrepancy},i}, \tag{9.40}$$

where $\mathbf{C}_{\rm obs}$ is the covariance matrix of the observational data, $\mathbf{C}_{\rm model,random}$ is the covariance matrix of the model arising from random noise (due to the finite number of trees simulated, and $\mathbf{C}_{{\rm discrepancy},i}$ is the covariance due to the $i^{\rm th}$ model discrepancy.

The model likelihood is then computed using:

$$\mathcal{L} = \frac{1}{\sqrt{(2\pi)^n |\mathbf{C}|}} \exp\left[-\frac{1}{2}\Delta \mathbf{C}^{-1} \Delta\right], \tag{9.41}$$

where $\Delta_i = \Phi_{{\rm model},i} - \Phi_{{\rm observed},i}$ is the difference between the model and observed mass functions, and n is the number of points in the mass function histogram.

When computing the Poisson term in the covariance matrix, the number of galaxies per bin is taken from data supplied by I. Baldry (private communication). The survey geometry and depth is described in §12.59.5.

[8] 7% arising from fiber collisions, 2% from failures in the Pymorph pipeline (M. Bernardi, private communication).

Finally, the completeness of the observational sample is estimated to be greater than 98% (P. Norberg, private communication). Therefore we add an additional contribution to the observed covariance matrix equal to $\mathcal{C}_{ij} = 0.02\phi_i\phi_j$ where ϕ is the observed mass function.

Tomczak et al. (2014) ZFOURGE Stellar Mass Functions

This constraint utilizes the stellar mass function for $0.2 < z < 3$ galaxies measured by Tomczak et al. [2014] from the ZFOURGE survey. The mass function, and its covariance matrix $\mathbf{C}_{\rm model}$, is calculated as described in §4.4.1, typically on-the-fly as Galacticus is run. If the `-modelDiscrepancies` option is given, then any multiplicative or additive discrepancies found are applied to the model mass function, and any additional covariance is added to the covariance matrix.

The covariance matrix is computed as

$$\mathbf{C} = \mathbf{C}_{\rm obs} + \mathbf{C}_{\rm model,random} + \sum_i \mathbf{C}_{{\rm discrepancy},i}, \tag{9.42}$$

where $\mathbf{C}_{\rm obs}$ is the covariance matrix of the observational data, $\mathbf{C}_{\rm model,random}$ is the covariance matrix of the model arising from random noise (due to the finite number of trees simulated, and $\mathbf{C}_{{\rm discrepancy},i}$ is the covariance due to the $i^{\rm th}$ model discrepancy.

The model likelihood is then computed using:

$$\mathcal{L} = \frac{1}{\sqrt{(2\pi)^n |\mathbf{C}|}} \exp\left[-\frac{1}{2}\Delta\mathbf{C}^{-1}\Delta\right], \tag{9.43}$$

where $\Delta_i = \Phi_{{\rm model},i} - \Phi_{{\rm observed},i}$ is the difference between the model and observed mass functions, and n is the number of points in the mass function histogram.

When computing the Poisson term in the covariance matrix, the number of galaxies per bin is taken directly from tables provided by R. Quadri (private communication).

Davidzon et al. (2013) VIPERS Stellar Mass Functions

This constraint utilizes the stellar mass function for $0.5 < z < 1.0$ galaxies measured by Davidzon et al. [2013] from the VIPERS survey. The mass function, and its covariance matrix $\mathbf{C}_{\rm model}$, is calculated as described in §4.4.1, typically on-the-fly as Galacticus is run. If the `-modelDiscrepancies` option is given, then any multiplicative or additive discrepancies found are applied to the model mass function, and any additional covariance is added to the covariance matrix.

The covariance matrix is computed as

$$\mathbf{C} = \mathbf{C}_{\rm obs} + \mathbf{C}_{\rm model,random} + \sum_i \mathbf{C}_{{\rm discrepancy},i}, \tag{9.44}$$

where $\mathbf{C}_{\rm obs}$ is the covariance matrix of the observational data, $\mathbf{C}_{\rm model,random}$ is the covariance matrix of the model arising from random noise (due to the finite number of trees simulated, and $\mathbf{C}_{{\rm discrepancy},i}$ is the covariance due to the $i^{\rm th}$ model discrepancy.

The model likelihood is then computed using:

$$\mathcal{L} = \frac{1}{\sqrt{(2\pi)^n |\mathbf{C}|}} \exp\left[-\frac{1}{2}\Delta\mathbf{C}^{-1}\Delta\right], \tag{9.45}$$

where $\Delta_i = \Phi_{{\rm model},i} - \Phi_{{\rm observed},i}$ is the difference between the model and observed mass functions, and n is the number of points in the mass function histogram.

When computing the Poisson term in the covariance matrix, the number of galaxies per bin is taken directly from tables supplied by I. Davidzon (private communication).

Muzzin et al. (2013) ULTRAVISTA Stellar Mass Functions

This constraint utilizes the stellar mass function for $0.2 < z < 4.0$ galaxies measured by Muzzin et al. [2013] from the ULTRAVISTA survey. The mass function, and its covariance matrix $\mathbf{C}_\mathrm{model}$, is calculated as described in §4.4.1, typically on-the-fly as GALACTICUS is run. If the `-modelDiscrepancies` option is given, then any multiplicative or additive discrepancies found are applied to the model mass function, and any additional covariance is added to the covariance matrix.

The covariance matrix is computed as

$$\mathbf{C} = \mathbf{C}_\mathrm{obs} + \mathbf{C}_\mathrm{model,random} + \sum_i \mathbf{C}_\mathrm{discrepancy},i, \tag{9.46}$$

where $\mathbf{C}_\mathrm{obs}$ is the covariance matrix of the observational data, $\mathbf{C}_\mathrm{model,random}$ is the covariance matrix of the model arising from random noise (due to the finite number of trees simulated, and $\mathbf{C}_\mathrm{discrepancy},i$ is the covariance due to the i^th model discrepancy.

The model likelihood is then computed using:

$$\mathcal{L} = \frac{1}{\sqrt{(2\pi)^n |\mathbf{C}|}} \exp\left[-\frac{1}{2}\Delta \mathbf{C}^{-1} \Delta\right], \tag{9.47}$$

where $\Delta_i = \Phi_\mathrm{model},i - \Phi_\mathrm{observed},i$ is the difference between the model and observed mass functions, and n is the number of points in the mass function histogram.

When computing the Poisson term in the covariance matrix, the completeness is taken to be 95% in the lowest mass bins, and 100% in all other bins [Muzzin et al., 2013].

Moustakas et al. (2013) PRIMUS Stellar Mass Functions

This constraint utilizes the stellar mass functions for $z \approx 0.2$ to $z \approx 1$ galaxies measured by Moustakas et al. [2013] from the PRIMUS survey. The mass functions, and their covariance matrices $\mathbf{C}_\mathrm{model}$, are calculated as described in §4.4.1, typically on-the-fly as GALACTICUS is run. If the `-modelDiscrepancies` option is given, then any multiplicative or additive discrepancies found are applied to the model mass function, and any additional covariance is added to the covariance matrix.

The covariance matrix is computed as

$$\mathbf{C} = \mathbf{C}_\mathrm{obs} + \mathbf{C}_\mathrm{model,random} + \sum_i \mathbf{C}_\mathrm{discrepancy},i, \tag{9.48}$$

where $\mathbf{C}_\mathrm{obs}$ is the covariance matrix of the observational data, $\mathbf{C}_\mathrm{model,random}$ is the covariance matrix of the model arising from random noise (due to the finite number of trees simulated, and $\mathbf{C}_\mathrm{discrepancy},i$ is the covariance due to the i^th model discrepancy.

The model likelihood is then computed using:

$$\mathcal{L} = \frac{1}{\sqrt{(2\pi)^n |\mathbf{C}|}} \exp\left[-\frac{1}{2}\Delta \mathbf{C}^{-1} \Delta\right], \tag{9.49}$$

where $\Delta_i = \Phi_\mathrm{model},i - \Phi_\mathrm{observed},i$ is the difference between the model and observed mass functions, and n is the number of points in the mass function histogram.

When computing the Poisson term in the covariance matrix, the number of galaxies per bin is taken directly from Moustakas et al. [2013]. The survey geometry and depth is described in §12.59.6.

Martin et al. (2010) ALFALFA HI Mass Function

This constraint utilizes the HI mass function for $z \approx 0.0$ galaxies measured by Martin et al. [2010] from the ALFALFA survey. The mass function, and its covariance matrix $\mathbf{C}_{\rm model}$, is calculated as described in §4.4.1, typically on-the-fly as Galacticus is run. If the `-modelDiscrepancies` option is given, then any multiplicative or additive discrepancies found are applied to the model mass function, and any additional covariance is added to the covariance matrix.

The covariance matrix is computed as

$$\mathbf{C} = \mathbf{C}_{\rm obs} + \mathbf{C}_{\rm model,random} + \sum_i \mathbf{C}_{{\rm discrepancy},i}, \tag{9.50}$$

where $\mathbf{C}_{\rm obs}$ is the covariance matrix of the observational data, $\mathbf{C}_{\rm model,random}$ is the covariance matrix of the model arising from random noise (due to the finite number of trees simulated, and $\mathbf{C}_{{\rm discrepancy},i}$ is the covariance due to the $i^{\rm th}$ model discrepancy.

The model likelihood is then computed using:

$$\mathcal{L} = \frac{1}{\sqrt{(2\pi)^n |\mathbf{C}|}} \exp\left[-\frac{1}{2}\Delta \mathbf{C}^{-1} \Delta\right], \tag{9.51}$$

where $\Delta_i = \Phi_{{\rm model},i} - \Phi_{{\rm observed},i}$ is the difference between the model and observed mass functions, and n is the number of points in the mass function histogram.

When computing the Poisson term in the covariance matrix, the number of galaxies per bin is taken directly from Fig. 5 of Martin et al. [2010] such that incompleteness is automatically accounted for. The survey geometry and depth is described in §12.59.8.

The resulting maximum likelihood mass function is shown in Fig. 9.5, clearly illustrating that this parametric HOD can produce an excellent match to the observed mass function. The resulting correlation matrix is shown in Fig. 9.6. As expected, at the higher masses the correlation matrix is dominated by the on-diagonal terms—arising from the Poisson fluctuations in the number of galaxies due to the scarcity of these massive systems. At lower masses the matrix has significant off-diagonal amplitude, indicating strong correlations between nearby bins, arising from both large-scale structure and halo contributions to the covariance. This structure significantly weakens the constraint arising from the low-mass end of the mass function.

Shen et al. (2003) Late-Type Galaxy Size Distribution

This constraint utilizes the distribution of Petrosian half-light radii for $z \approx 0.07$ late-type galaxies measured by Shen et al. [2003] from the SDSS. The size function reported by Shen et al. [2003] is converted to the appropriate cosmology for the given Galacticus model (assuming that sizes scale as the angular diameter distance, and masses as the square of the luminosity distance).

Given a Galacticus model, total stellar masses of model galaxies are adjusted using:

$$M_\star \rightarrow \mathbf{GS} M_\star \tag{9.52}$$

where the $\mathbf{S}$ operator is a multiplicative factor accounting for systematic errors in stellar mass determination and is equal to [Behroozi et al., 2010]

$$\log_{10} S = \mu + \kappa \log_{10}\left(\frac{M_\star}{10^{11.0} M_\odot}\right) \tag{9.53}$$

where $\mu =$ `[diskGalaxySizesSDSSZ0.07MassSystematic0]`, $\kappa =$ `[diskGalaxySizesSDSSZ0.07MassSystematic1]`, and the $\mathbf{G}$ operator is a multiplicative factor drawn from a log-normal distribution of width 0.0806 dex

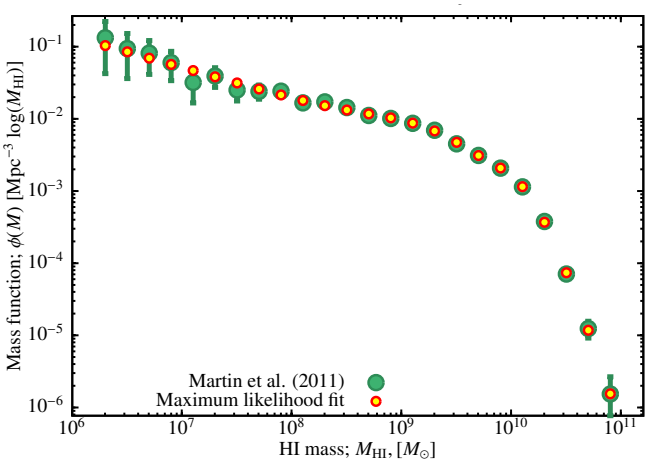

Figure 9.5.: The maximum likelihood mass function obtained from our parametric HOD (yellow points), compared to the observed HI mass function of Martin et al. (2010; green points).

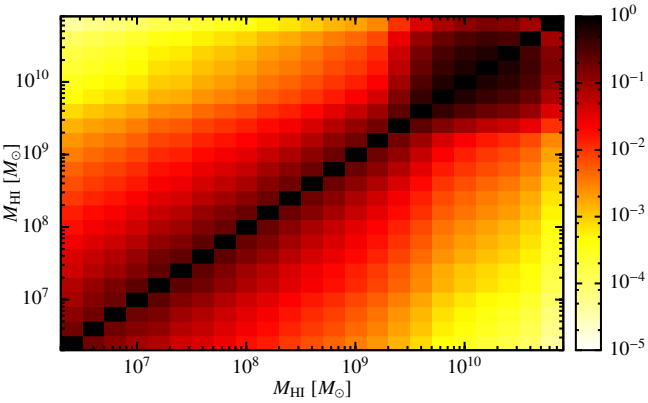

Figure 9.6.: The correlation matrix of the observed galaxy HI mass function of Martin et al. [2010]. Color indicates the strength of correlation between bins, according to the scale shown on the right.

for each galaxy to mimic the effects of random errors on stellar masses (motivated by the statement from Shen et al. [2003] who quote the 95% confidence interval on masses as being ±40%).

Note: This analysis currently assumes that model galaxies have disk Petrosian half-mass radii of

$$R_{50} = 1.6676 \frac{1}{\sqrt{2}} \lambda R_{\rm vir}. \tag{9.54}$$

Disk sizes of model galaxies are then adjusted using:

$$R_{50} \rightarrow \mathbf{G}\mathbf{S} R_{50} \tag{9.55}$$

where the $\mathbf{S}$ operator is a multiplicative factor accounting for systematic errors in radius determination and in determination of radius from halo virial radius and spin and is equal to

$$\log_{10} S = \mu + \kappa \log_{10} \left(\frac{R_{50}}{1 \mathrm{kpc}} \right) \tag{9.56}$$

where $\mu =$ `[iskGalaxySizesSDSSZ0.07RadiusSystematic0]`, $\kappa =$ `[iskGalaxySizesSDSSZ0.07RadiusSystematic1]`, and the $\mathbf{G}$ operator is a multiplicative factor drawn from a log-normal distribution of width 0.0128 dex for each galaxy to mimic the effects of random errors on disk radii (estimated from the fractional errors reported in the SDSS database).

The model sizes and masses are then used to construct a mass-dependent radius function by binning into a 2-D histogram using the size and mass bins reported by Shen et al. [2003] (modified as described above) as the centers of the bins (with bin boundaries placed at the geometric means of consecutive bin centers).

If the `-modelDiscrepancies` option is given, then any multiplicative or additive discrepancies found are applied to the model mass function, and any additional covariance is added to the covariance matrix.

The covariance matrix is computed as

$$\mathbf{C} = \mathbf{C}_{\rm obs} + \mathbf{C}_{\rm model,random} + \sum_i \mathbf{C}_{{\rm discrepancy},i}, \tag{9.57}$$

where $\mathbf{C}_{\rm obs}$ is the covariance matrix of the observational data, $\mathbf{C}_{\rm model,random}$ is the covariance matrix of the model arising from random noise (due to the finite number of trees simulated, and $\mathbf{C}_{{\rm discrepancy},i}$ is the covariance due to the $i^{\rm th}$ model discrepancy.

The model covariance matrix is estimated using the sample methods as described in §9.5.3. The only difference is that in this case we have a 2-D histogram. This 2-D histogram is "flattened" into a 1-D vector for purposes of likelihood computation however, so covariance matrix estimation proceeds unchanged. (Note that correlations between mass bins are accounted for, in additional to correlations between radius bins.) Since the radius functions of Shen et al. [2003] are normalized to unity at each mass, we must account for this in the covariance matrix. The radius function transforms as:

$$f_{ik} \rightarrow \frac{f_{ik}}{\Delta \log_{10} R \sum_i f_{ik}}, \tag{9.58}$$

where i indexes radius bins, k indexes mass bins, and $\Delta \log_{10} R$ is the width of the radius bin. The Jacobian of this transformation is simply

$$J_{ij} = \frac{\delta_{ij} - f_i}{\Delta \log_{10} R \sum_i f_{ik}}. \tag{9.59}$$

Therefore, the covariance matrix is modified according to $\mathcal{C} \rightarrow J \mathcal{C} J^{\rm T}$. The same transformation is applied to the covariance matrix of the observed data (for which the reported errors are simply the Poisson errors on each bin).

The model likelihood is then computed using:

$$\mathcal{L} = \frac{1}{\sqrt{(2\pi)^n |\mathbf{C}|}} \exp\left[-\frac{1}{2}\Delta\mathbf{C}^{-1}\Delta\right], \tag{9.60}$$

where $\Delta_i = \Phi_{\mathrm{model},i} - \Phi_{\mathrm{observed},i}$ is the difference between the model and observed mass functions, and n is the number of points in the mass function histogram.

9.6. Constraint Compilations

To specify which constraints will be applied to a particular model, a compilation file is used. These must be stored in `constraints/compilations`. An example of such a file follows:

```
<constraintCompilation>
  <constraint>
    <definition>constraints/constraints/stellarMassFunction_SDSS_z0.07.xml</definition>
    <weight>1.0</weight>
  </constraint>
  <constraint>
    <definition>constraints/constraints/hiMassFunction_ALFALFA_z0.00.xml</definition>
    <weight>1.0</weight>
  </constraint>
</constraintCompilation>
```

Each `constraint` element specifies one constraint that will be included in this compilation, and must contain a `definition` element, giving the path of the configuration file for this constraint, and a `weight` element which allows the relative weight given to each constraint to be varied[9].

9.7. Constraint File

Galacticus has a complete constraints infrastructure which implements various MCMC algorithms to analyze the posterior probability distribution of the model given some compilation of constraints. The infrastructure is message passing interface (MPI) parallelized and ideal for running on large compute clusters.

To perform a constraint calculation simply build the constraint code:

```
make Constrain_Galacticus.exe
```

and run with a parameter file and configuration file. Typically, you will want to run this code under MPI, for example:

```
mpirun -n 4 Constrain_Galacticus.exe mcmcParameters.xml mcmcConfig.xml
```

would run 4 processes (typically you will need to run many more than this). If running on a portable batch system (PBS) queue, embed this command in a suitable PBS script and submit.

The parameter file follows the same format as a standard Galacticus parameter file and specifies the values of parameters to be used. For example, the seed used fo pseudo-random number sequences can be specified in this file.

The configuration file specifies the details of the constraint simulation to be performed. An example configuration file is:

[9] Note that, if your constraints are computing correct likelihoods, re-weighting them may not be a good idea. *Caveat constrainor.*

```
<?xml version="1.0" encoding="UTF-8"?>
<simulationConfig>

  <likelihood>
    <type>Galacticus</type>
    <name>verySimplisticToStellarMassFunction</name>
    <compilation>stellarMassFunction_SDSS_z0.07.xml</compilation>
    <baseParameters>./mcmcWork/verySimplisticToStellarMassFunctionBase.xml</baseParameters>
    <workDirectory>./mcmcWork</workDirectory>
    <scratchDirectory>./mcmcScratch</scratchDirectory>
    <report>no</report>
    <randomize>no</randomize>
    <threads>4</threads>
    <saveState>no</saveState>
    <cpulimit>1200</cpulimit>
    <coredump>NO</coredump>
    <coredumpsize>0</coredumpsize>
    <sequentialModels>no</sequentialModels>
    <memoryLimit>2gb</memoryLimit>
    <environment>LD_LIBRARY_PATH=/opt/gcc-trunk/lib:/opt/gcc-trunk/lib64:/usr/local/upstream/lib:$LD_LI
    <environment>PATH=/opt/gcc-trunk/bin:$PATH</environment>
  </likelihood>

  <convergence>
    <type>GelmanRubin</type>
    <Rhat>1.2</Rhat>
    <burnCount>100</burnCount>
    <testCount>100</testCount>
    <outlierCountMaximum>0</outlierCountMaximum>
    <outlierSignificance>0.95</outlierSignificance>
    <outlierLogLikelihoodOffset>60</outlierLogLikelihoodOffset>
  </convergence>

  <state>
    <type>history</type>
    <acceptedStateCount>100</acceptedStateCount>
  </state>

  <proposalSize>
    <type>adaptive</type>
    <gammaInitial>1.77</gammaInitial>
    <gammaFactor>1.414</gammaFactor>
    <acceptanceRateMinimum>0.4</acceptanceRateMinimum>
    <acceptanceRateMaximum>0.6</acceptanceRateMaximum>
    <updateCount>10</updateCount>
  </proposalSize>

  <randomJump>
    <type>adaptive</type>
```

```
  </randomJump>

  <simulation>
    <type>temperedDifferentialEvolution</type>
    <stepsMaximum>1000000</stepsMaximum>
    <stepsPostConvergence>100000</stepsPostConvergence>
    <acceptanceAverageCount>100</acceptanceAverageCount>
    <logFileRoot>./mcmcWork/mcmc/chains</logFileRoot>
    <temperatureMaximum>64.0</temperatureMaximum>
    <untemperedStepCount>20</untemperedStepCount>
    <temperedLevels>10</temperedLevels>
    <stepsPerLevel>10</stepsPerLevel>
    <logFlushCount>10<logFlushCount>
  </simulation>

  <parameters>
    <parameter>
      <name>starFormationTimescaleDisksHaloScalingVirialVelocityExponent</name>
      <prior>
<distribution>
  <type>uniform</type>
  <minimum>-6.0</minimum>
  <maximum>+0.0</maximum>
</distribution>
      </prior>
      <mapping>
        <type>linear</type>
      </mapping>
      <random>
<type>Cauchy</type>
<median>0.0</median>
<scale>0.006</scale>
      </random>
    </parameter>
    <parameter>
      <name>starFormationTimescaleDisksHaloScalingRedshiftExponent</name>
      <prior>
<distribution>
  <type>uniform</type>
  <minimum>-1.0</minimum>
  <maximum>+4.0</maximum>
</distribution>
      </prior>
      <mapping>
        <type>linear</type>
      </mapping>
      <random>
<type>Cauchy</type>
<median>0.0</median>
```

```
<scale>0.005</scale>
      </random>
    </parameter>
  </parameters>

</simulationConfig>
```

The following subsections describe each entry in this file.

9.7.1. likelihood

The likelihood section specifies the likelihood function to be used in the simulation. The type of likelihood to use is specified by the type element. The available choices are described in the following subsections.

multivariateNormal

The likelihood is a simple multivariate Gaussian, intended primarily for testing purposes. The distribution parameters are specified within the likelihood element using:

```
<mean>0.45 0.50</mean>
<covariance>
  <row>1.0e-4 -0.9e-4</row>
  <row>-0.9e-4 1.0e-4</row>
</covariance>
```

where the mean element gives the mean vector of N elements, and the covariance element contains N row elements each containing a vector of N elements giving a single row of the covariance matrix. The likelihood is then:

$$\log \mathcal{L} = -\frac{1}{2}\Delta\mathcal{C}^{-1}\Delta^{\mathrm{T}}, \tag{9.61}$$

where $Delta = \theta - \bar{\theta}$, θ is the state, $\bar{\theta}$ is the mean, and $\mathcal{C}$ is the covariance matrix.

multivariateNormalStochastic

The likelihood is identical to that of the multivariateNormal class, except that the likelihood function is evaluated stochastically. In addition to the parameter of the multivariateNormal class, two additional parameters are required and are specified within the likelihood element using:

```
<realizationCount>4000</realizationCount>
<realizationCountMinimum>10</realizationCountMinimum>
```

When evaluating the likelihood, the state vector is set equal to

$$S_i' = \sum_{j=1}^{N} \frac{2U(S_i)}{N}, \tag{9.62}$$

where $N =$realizationCount and $U(x)$ is a uniform random deviate in the range 0 to x. This results in a variance in S_i' of $S_i^2/3N$. This variance is added to the covariance used in evaluating the likelihood. When evaluating the likelihood at a higher temperature the number of realizations is reduced (which increases the covariance, which has the same effect as increasing the temperature) to speed computation, and the likelihood corrected for this fact. The number of realizations is reduced to N/T, but never allowed to fall below realizationCountMinimum.

Galacticus

The likelihood is computed by running and analyzing a Galacticus model. The details of the model to run are specified by the follow content within the `likelihood` element:

```
<name>verySimplisticToStellarMassFunction</name>
<compilation>stellarMassFunction_SDSS_z0.07.xml</compilation>
<baseParameters>./mcmcWork/verySimplisticToStellarMassFunctionBase.xml</baseParameters>
<workDirectory>./mcmcWork</workDirectory>
<scratchDirectory>./mcmcScratch</scratchDirectory>
<report>no</report>
<randomize>no</randomize>
<threads>4</threads>
<saveState>no</saveState>
<cpulimit>1200</cpulimit>
<memoryLimit>2gb</memoryLimit>
<environment>LD_LIBRARY_PATH=/opt/gcc-trunk/lib:/opt/gcc-trunk/lib64:/usr/local/upstream/lib:$LD_LIBRA
<environment>PATH=/opt/gcc-trunk/bin:$PATH</environment>
<environment>GFORTRAN_ERROR_DUMPCORE=NO</environment>
<treesPerDecadeMinimum></treesPerDecadeMinimum>
```

The entries have the following meanings:

`name` A name to use for this calculation. It will be used as the name for jobs submitted to the PBS queue for example.

`compilation` Specifies the compilation file to be used for this analysis.

`baseParameters` Specifies the path to a Galacticus parameter file which will be used as the base set of parameter on top of which any parameter variations will be applied.

`report` If set to `yes`, reports additional debugging information during the run.

`scratchDirectory` The full path to a scratch directory where Galacticus model outputs and other temporary data will be written.

`memoryLimit` A memory limit to be passed to PBS.

`randomize` If `yes` then each model evaluation will be performed with a different random number seed. Otherwise, the same seed is used in all cases. Experiment shows that changing the random number seed between evaluations can seriously limit the ability of MCMC algorithms to converge.

`threads` The number of parallel OpenMP threads to use for each Galacticus model. It is recommended that this be set to the number of available cores on each node, and semaphoring (see §sec:Semaphores) be used. In this way, each copy of Galacticus on a node will share resources, but as one instance finishes, the others will be able to make use of the freed resources.

`coredump` If present, this value will be assigned to the environment variable `GFORTAN_ERROR_DUMPCORE` to allow core dumps on error conditions to be switched on or off.

`coredumpsize` If present, limit the size of core dumps to this value (a value of `unlimited` allows for unlimited size core dumps.

`sequentialModels` If set to `yes` the Galacticus models are run sequentially on each node, rather than all at once.

`saveState` If `yes` then GALACTICUS will save its internal state prior to beginning evolution of each merger tree. This is intended for debugging purposes and so should normally be set to `no`.

`threads` The total number of threads (i.e. parallel chains) to use.

`environment` One of more such element can appear. Each specifies the value of an environment variable to be set prior to launching the GALACTICUS model.

`useFixedTrees` If set to `yes`, then a fixed set of merger trees are used for all model evaluations. This fixed set is generated automatically as needed[10], using the base parameters.

`fixedTreesInScratch` If set to `yes`, any fixed set of merger trees generated will be stored in the scratch directory, otherwise they will be stored in the work directory. (Storing in scratch can often be faster, particularly if `/dev/shm` is used as scratch, but for large sets of trees the available scratch storage space may be insufficient.)

When evaluating model likelihood at temperatures above unity, GALACTICUS will attempt to speed computation by reducing the number of trees simulated. Specifically, it makes use of the facts that the model covariance, $\mathcal{C} \propto N_\mathrm{tree}^{-1}$, and that the model log-likelihood, $\mathcal{L} \propto \mathcal{C}^{-1}/T$. Therefore, increasing the temperature is equivalent to decreasing the number of trees simulated by the same proportion. GALACTICUS therefore runs a simulation with a number of trees per decade of halo mass equal to `int([mergerTreeBuildTreesPerDecade]/T)` or the value of the `treesPerDecadeMinimum` element, whichever is smaller. This defines an effective temperature $T' = N'_\mathrm{trees}/N_\mathrm{trees}$ where N_trees is the original number of trees per decade, and N'_trees is the reduced number.

The model likelihood, $\mathcal{L}'$ is then evaluated using this reduced number of trees. The covariance matrix of the data is increased by a factor T' when evaluating the likelihood, as are any additions to the covariance due to model discrepancy. Finally, the likelihood at $T = 1$ is computed using:

$$\log \mathcal{L} = T' \log \mathcal{L}' + (T' - 1) \left[\frac{N}{2} \log 2\pi + \frac{1}{2} \log |\mathcal{C}'| \right], \tag{9.63}$$

where N is the dimension of the covariance matrix and $\mathcal{C}'$ is the full covariance matrix.

gaussianRegression

The likelihood is computed either using another likelihood function (the "simulator") or via Gaussian regression emulation of that simulator. The details of the emulation algorithm are specified by the follow content within the `likelihood` element:

```
<emulatorRebuildCount>100</emulatorRebuildCount>
<polynomialOrder>2</polynomialOrder>
<sigmaBuffer>3.0</sigmaBuffer>
<logLikelihoodBuffer>10.0</logLikelihoodBuffer>
<logLikelihoodErrorTolerance>0.01</logLikelihoodErrorTolerance>
<reportCount>100</reportCount>
<simulatorLikelihood>
 .
 .
 .
</simulatorLikelihood>
```

[10] If the temperature is being varied the number of trees per decade of halo mass changes throughout the simulation. In this case, a new set of trees is generated for each distinct value of the numbers of trees per decade.

The entries have the following meanings:

`emulatorRebuildCount` The number of simulator evaluations from which the emulator is built;

`polynomialOrder` The order of the polynomial fitted to the simulator likelihoods prior to Gaussian regression;

`sigmaBuffer` See below;

`logLikelihoodBuffer` See below;

`logLikelihoodErrorTolerance` See below;

`reportCount` The number of likelihood evaluations between successive reports on the status of the emulator;

`emulateOutliers` If true, then outlier chains are always emulated post-convergence (this is safe if such chains are not used in constructing proposals for non-outlier chains);

`simulatorLikelihood` Contains another likelihood function definition which will be used to construct the simulator.

In detail, this likelihood function first collects `emulatorRebuildCount` likelihood evaluations from the simulator. It then fits a polynomial of order `polynomialOrder` and of dimension equal to the dimension of the state vector to the simulated likelihoods. Gaussian regression is performed on the residuals of the simulated likelihoods after this polynomial fit is removed. Once the emulator has been built in this way every second simulated state is discarded, and accumulation of new simulated states continues. Once `emulatorRebuildCount` simulated states have once again been accumulated a new simulator is built. This ensures that the emulator does not lose all information used in building the previous emulator[11], instead information from older emulators decays exponentially.

Once an emulator has been built, on each successive likelihood evaluation the emulated log-likelihood $\log \mathcal{L}_\mathrm{e}$ and its error estimate $\sigma_{\log \mathcal{L}_\mathrm{e}}$ are computed. The emulated likelihood is then returned if:

$$\log \mathcal{P}' + \log \mathcal{L}_\mathrm{e} + N \sigma_{\log \mathcal{L}_\mathrm{e}} < \log \mathcal{P} + \log \mathcal{L} - T \Delta \log \mathcal{L}, \tag{9.64}$$

where $N =$`sigmaBuffer`, $\Delta \log \mathcal{L} =$`logLikelihoodBuffer`, T is the temperature, $\log \mathcal{L}$ is the current log-likelihood, $\log \mathcal{P}$ is the current log-prior probability, and $\log \mathcal{P}'$ is the proposed log-prior probability, or if

$$\sigma_{\log \mathcal{L}_\mathrm{e}} < T \sigma_{\log \mathcal{L}}, \tag{9.65}$$

where $\sigma_{\log \mathcal{L}} =$`logLikelihoodErrorTolerance`, otherwise the simulator is used to compute the exact likelihood. In this way, the emulated likelihood is used if it is sufficiently below the current likelihood that, even accounting for the emulation error, transition to the new state is highly unlikely, or if the error on the likelihood emulation is sufficiently small that it will not have a significant effect on the transition probabilty to the proposed state.

If verbosity is set to 2 or greater than a report will be issued every `reportCount` evaluations. The report will give the proportions of simulated vs. emulated evaluations. Additionally, during the evaluation where the report is issued, both the emulated and simulated log-likelihoods are evaluated and are tested to see if they lie within $3\sigma_{\log \mathcal{L}_\mathrm{e}}$ of each other. The rate of failures (i.e. where the two differ by more than this amount) is then reported.

[11] This would be unfortunate as the second emulator to be built would then contain information on only those regions of the state space that were poorly emulated before.

posteriorPrior

The likelihood is computed either using another likelihood function (the "wrapped" likelihood), while including in the likelihood an esimate of the posterior probability of a previous simulation. This effectively allows the posterior of the previous simulation to be used as a prior on the current simulation. The details of the likelihood are specified by the follow content within the `likelihood` element:

```
<chainBaseName>./oldChains</chainBaseName>
<neighborCount>32</neighborCount>
<tolerance>1.0e-3</tolerance>
<wrappedLikelihood>
 .
 .
 .
</wrappedLikelihood>
```

The entries have the following meanings:

`chainBaseName` The base name for the old set of MCMC chains to use as the new prior;

`neighborCount` The number of neighbor points to use in kernel density estimation of the posterior probability;

`tolerance` Tolerance used in finding nearest neighbors;

`wrappedLikelihood` Contains another likelihood function definition which will be used to provide the current likelihood.

This method uses the Approximate Nearest Neighbor library to locate `neightborCount` nearest neighbor points in the set of converged states found in the given chains. The `tolerance` element determines the accuracy of nearest neighbor finding (see the Approximate Nearest Neighbor documentation for details).When finding nearest neighbors in the MCMC chains, parameters are mapped using whatever mappings are currently active, and distances in each dimension (as used in the metric to determine nearest neighbors) are scaled by the root-variance in that parameter in the converged MCMC chains. The posterior likelihood of the MCMC chains is then estimated from the nearest neighbors using kernel density estimation with a Gaussian kernel with bandwidth equal to the distance to the furthest of the nearest neighbors.

massFunction

The likelihood is computed as

$$\log \mathcal{L} = -\frac{1}{2}\Delta \cdot \mathcal{C}^{-1} \cdot \Delta^{\mathrm{T}}, \tag{9.66}$$

where $\mathcal{C}$ is the covariance matrix, and $\Delta_i = w_i^{\mathrm{model}} - w_i^{\mathrm{obs}}$, w_i^{model} is the computed mass function at the i^{th} separation, and w_i^{obs} is the observed mass function at the i^{th} separation. The mass function is computed using the halo model and the parameterized conditional galaxy mass function of [Behroozi et al., 2010, see also Leauthaud et al. [2011]; §12.8.1]. The details of the mass function calculation are specified by the follow content within the `likelihood` element:

```
<haloMassMinimum>1.0e9</haloMassMinimum>
<haloMassMaximum>1.0e15</haloMassMaximum>
<redshiftMinimum>0.0</redshiftMinimum>
<redshiftMaximum>0.5</redshiftMaximum>
<massFunctionFileName>projectedCorrelation.hdf5</massFunctionFileName>
```

The entries have the following meanings:

`haloMass(Min|Max)imum` The minimum/maximum halo mass over which to integrate in the halo model;

`redshift(Min|Max)imum` The minimum/maximum redshift over which to integrate in the halo model;

`massFunctionFileName` The name of an HDF5 file containing the observed mass function and its covariance matrix.

The HDF5 file specified by the `massFunctionFileName` element should contain a `mass` dataset, giving the masses at which the mass function is measured (in units of $M_\odot$), a `massFunctionObserved` dataset giving the observed values of the mass function at those masses (in units of Mpc^{-3} per $\log M$), and a `covariance` dataset, giving the covariance of the mass function (in units of Mpc^{-6}).

projectedCorrelationFunction

The likelihood is computed as

$$\log \mathcal{L} = -\frac{1}{2} \Delta \cdot \mathcal{C}^{-1} \cdot \Delta^{\mathrm{T}}, \tag{9.67}$$

where $\mathcal{C}$ is the covariance matrix, and $\Delta_i = w_i^{\mathrm{model}} - w_i^{\mathrm{obs}}$, w_i^{model} is the computed projected correlation function at the i^{th} separation, and w_i^{obs} is the observed projected correlation function at the i^{th} separation. The projected correlation function is computed using the halo model and the parameterized conditional galaxy mass function of [Behroozi et al., 2010, see also Leauthaud et al. [2011]; §12.8.1]. The details of the projected correlation function calculation are specified by the follow content within the `likelihood` element:

```
<haloMassMinimum>1.0e9</haloMassMinimum>
<haloMassMaximum>1.0e15</haloMassMaximum>
<redshiftMinimum>0.0</redshiftMinimum>
<redshiftMaximum>0.5</redshiftMaximum>
<projectedCorrelationFunctionFileName>projectedCorrelation.hdf5</projectedCorrelationFunctionFileName>
```

The entries have the following meanings:

`haloMass(Min|Max)imum` The minimum/maximum halo mass over which to integrate in the halo model;

`redshift(Min|Max)imum` The minimum/maximum redshift over which to integrate in the halo model;

`projectedCorrelationFunctionFileName` The name of an HDF5 file containing the observed projected correlation function and its covariance matrix.

The HDF5 file specified by the `projectedCorrelationFunctionFileName` element should contain a `separation` dataset, giving the spearations at which the projected correlation function is measured (in units of Mpc), a `projectedCorrelationFunctionObserved` dataset giving the observed values of the projected correlation function at those separations (in units of Mpc), and a `covariance` dataset, giving the covariance of the projected correlation function (in units of Mpc^2).

9.7.2. convergence

The `convergence` section specifies the criterion to be used to judge when the simulation has converged. The type of convergence criterion to use is specified by the `type` element. The available choices are described in the following subsections.

`never`

This option assumes that the simulation never converges, and so the calculation will run indefinitely. It is intended primarily for testing purposes.

`GelmanRubin`

This option adopts the convergence criterion proposed by Gelman and Rubin (1992; see also Brooks and Gelman 1998), which compares the variance in parameter values within chains to that between chains. Outlier detection is applied to the chains using a standard Grubb's outlier test. The behavior of this criterion is controlled by the following options which should be placed within the `convergence` element:

`Rhat` The correlation coefficient, $\hat{R}$, value at which to declare convergence.

`burnCount` Set number of steps to burn before applying the convergence test.

`testCount` Set the number of steps between successive applications of the convergence test.

`outlierSignificance` The significance level required in outlier detection.

`outlierLogLikelihoodOffset` The offset in log-likelihood from the current maximum likelihood chain required for a chain to be declared to be an outlier.

`outlierCountMaximum` The maximum number of outlier chains allowed.

9.7.3. `state`

The `state` section specifies the type of object used to record the state of the simulation. The type of state object to use is specified by the `type` element. The available choices are described in the following subsections.

`simple`

This type stores the current state but makes no attempt to record a history of the state and so cannot provide measures of the mean or variance of state over the simulation history. It does, however, maintain a running average of the state acceptance rate. The number of steps over which the acceptance rate should be computed is specified by the `acceptedStateCount`.

`history`

An extension of the `simple` state, this type also records the mean and variance of each parameter over the history of the simulation.

`correlation`

An extension of the `history` state, this type also computes and stores the correlation length in each parameter (which is taken to be the median correlation length over all non-outlier chains).

9.7.4. `stoppingCriterion`

The `stoppingCriterion` section specifies the type of object used to determine when the simulation should be stopped. The type of stopping criterion object to use is specified by the `type` element. The available choices are described in the following subsections.

`never`

This type will cause the simulation to never stop.

`stepCount`

This type will cause the simulation to stop when at least a number of steps (as specified in the `stopAfterCunt` element) have accrued post-convergence.

`correlationLengthCount`

This type will cause the simulation to stop when at least a number of correlation lengths (as specified in the `stopAfterCunt` element) have accrued post-convergence.

9.7.5. `stateInitializor`

The `stateInitializor` section specifies the type of object used to initialize the state of chains at the beginning of the simulation. The type of state initializor object to use is specified by the `type` element. The available choices are described in the following subsections.

`priorRandom`

This type simply draws a random state vector from the prior distributions of the parameters.

`resume`

This type resumes from a previous simulation by setting the chain states to the states at the end of that simulation. The `logFileRoot` element is used to specify the log-file root name used in the previous simulation.

`latinHypercube`

This type uses a Latin hypercube design (in the cumulative probability distribution of each parameter) to assign initial state vectors. In particular, a maximin design is used in which a number of trial Latin hypercubes are constructed and the hypercube with the greatest minimum distance between any pair of state vectors is selected. The `maximinTrialCount` element is used to specify the number of trial hypercubes to construct.

9.7.6. `proposalSize`

The `proposalSize` section specifies the method to use when selecting the proposal size parameter, γ (the fraction of the vector connecting to chain state to be used as the proposal for another chain), for use in differential evolution simulations. The proposal size algorithm to use is specified by the `type` element. The available choices are described in the following subsections.

`fixed`

This option uses a fixed γ specified by the `gamma` element.

`adaptive`

This option adaptively changes γ in an attempt to maintain the acceptance rate at an acceptable level. The algorithm is controlled by the following parameters (to be specified as elements within the `proposalSize` element):

`gammaInitial` The initial value for γ.

`gammaFactor` The multiplicative factor by which γ should be increased or decreased if the acceptance rate is out of range.

`gammaMinimum` The smallest value allowed for γ.

`gammaMaximum` The largest value allowed for γ.

`acceptanceRateMinimum` The minimum acceptance rate to accept before reducing γ.

`acceptanceRateMaximum` The maximum acceptance rate to accept before reducing γ.

`updateCount` The number of steps between successive checks of the acceptance rate.

9.7.7. `proposalSizeTemperatureExponent`

The `proposalSizeTemperatureExponent` section specifies the method to use when selecting the exponent, α, for the temperature scaling of the proposal size parameter, γ (the fraction of the vector connecting to chain state to be used as the proposal for another chain), for use in tempered differential evolution simulations. The proposal size temperature exponent algorithm to use is specified by the `type` element. The available choices are described in the following subsections.

`fixed`

This option uses a fixed α specified by the `alpha` element.

`adaptive`

This option adaptively changes α in an attempt to maintain the gradient of the acceptance rate with the logarithm of temperature, $\mathrm{d}R/\mathrm{d}\ln T$, at an acceptable level. The algorithm is controlled by the following parameters (to be specified as elements within the `proposalSizeTemperatureExponent` element):

`exponentInitial` The initial value for α;

`exponentFactor` The additive factor by which α should be increased or decreased if the acceptance rate gradient is out of range;

`exponentMinimum` The smallest value allowed for α;

`exponentMaximum` The largest value allowed for α;

`acceptanceRateMinimum` The minimum acceptance rate gradient to accept before reducing α;

`acceptanceRateMaximum` The maximum acceptance rate gradient to accept before reducing α;

`updateCount` The number of steps between successive checks of the acceptance rate gradient.

9.7.8. randomJump

The `randomJump` section specifies the method to use when adding a random jump component to proposals in differential evolution simulations. The random jump algorithm to use is specified by the `type` element. The available choices are described in the following subsections.

simple

The random jumps are drawn directly from the distributions specified in the `random` element of each parameter (see §9.7.10).

adaptive

The random jumps are drawn from the distributions specified in the `random` element of each parameter (see §9.7.10) and then multiplied by the currently occupied range of each parameter (i.e. the maximum value of the parameter over all current chain states minus the minimum value of each parameter over all current chain states).

9.7.9. simulation

The `simulation` section specifies the algorithm to use to perform the simulation. The simulation algorithm to use is specified by the `type` element. The available choices are described in the following subsections.

differentialEvolution

This option uses the differential evolution algorithm of Terr Braak [2006]. Multiple, parallel chains are run and proposals are constructed by selecting two chains at random, taking a fraction, γ, of the vector connecting the two chain states and adding this to the state of the current chain. The details of the algorithm are controlled by the following elements which should be embedded within the `simulation` element:

`stepsMaximum` The maximum number of steps to take.

`acceptanceAverageCount` The number of steps over which to average the acceptance rate.

`stateSwapCount` The number of steps after which to set $\gamma = 1$ to allow chains to swap states.

`logFileRoot` The full path and root name of a file to log results to. The actual file name will have the rank of the MPI process appended to it.

`sampleOutliers` If set to `false` then proposals for non-outlier chains post-convergence are constructed only from other non-outlier chains. Otherwise, proposals for non-outleir chains post-convergence are constructed from all other chains.

simulatorStochasticDifferentialEvolution

This option extends the `differentialEvolution` option to run chains at a temperature matched to the uncertainty in the log-likelihood. This is designed to work with stochastic likelihood functions where an estimate of the uncertainty in the log-likelihood is available, and prevents the chains from becoming trapped in local maxima arising purely from random fluctuations. In addition to the options for the `differentialEvolution` algorithm, the details of the algorithm are controlled by the following elements whichy should be embedded within the `simulation` element:

`temperatureScale` The temperature scaling factor, *alpha*, described below.

In computing the acceptance probability for transitions between states, the chain temperature is set to

$$T = 1 + \alpha C \sqrt{\sigma^2_{\text{current}} + \sigma^2_{\text{proposed}}}, \tag{9.68}$$

where $C = \log(\hat{R}/\hat{R}_t)$, $\hat{R}$ is the current maximum (across all parameters) Gelman-Rubin convergence statistic, $\hat{R}_t$ is the target Gelman-Rubin convergence statistic at which convergence will be declared, and $\sigma^2_{\text{current}}$ and $\sigma^2_{\text{proposed}}$ are the variances in the log-likelihood of the current and proposed states respectively. This form ensures that the temperature declines to unity once the chains are converged (such that they will sample from the true posterior distribution), while ensuring that T is of order the size of the expected random fluctuations in the difference in log-likelihoods between states prior to chain convergence.

`temperedDifferentialEvolution`

This option extends the `differentialEvolution` option to include tempering during which the likelihood function is heated up and cooled down to allow chains to more easily walk through the likelihood landscape. In addition to the options for the `differentialEvolution` algorithm, the details of the algorithm are controlled by the following elements whichy should be embedded within the `simulation` element:

`untemperedStepCount` The number of untempered (i.e. $T = 1$) steps to take between tempering cycles.

`temperatureMaximum` The maximum temperature to use when tempering.

`temperedLevels` The number of tempered levels to use.

`stepsPerLevel` The number of differential evolution steps to take at each tempering level.

`logFlushCount` The number of steps after which the log file will be flushed to disk.

In each tempering cycle, the temperature is raised through levels $1 \ldots N$ (where $N =$`temperedLevels`), and then back down through levels $N - 1 \ldots 1$. The temperature at level i is given by:

$$\log T_i = \frac{i}{N} \log T_{\max}, \tag{9.69}$$

where $T_{\max} =$`temperatureMaximum`. During tempered steps, the γ parameter of the differential evolution algorithm is increased by a factor T^α, where α is provided by the `proposalSizeTemperatureExponent` class. A value of $\alpha = 1/2$ is optimal for a Gaussian likelihood.

`annealedDifferentialEvolution`

This option extends the `differentialEvolution` option to include an annealing schedule—the simulation begins at high temperature, waits for convergence, lowers the temperature and repeats until convergence at $T = 1$ is reached. In addition to the options for the `differentialEvolution` algorithm, the details of the algorithm are controlled by the following elements whichy should be embedded within the `simulation` element:

`temperatureMaximum` The maximum temperature to use when tempering.

`temperatureLevels` The number of temperature levels to use.

The temperature at level i is given by:

$$\log T_i = \frac{i-1}{N-1} \log T_{\max}, \tag{9.70}$$

where $T_{\max} =$`temperatureMaximum` and $N =$`temperatureLevels`.

9.7.10. Parameters and Priors

The `parameters` section contains a list of all parameters to be varied in the analysis. Each parameter is described by one `parameter` element. That element must contain a `name` element, which gives the name of the parameter, a `prior` element that contains a `distribution` element defining the distribution for this prior, a `mapping` element that describes the mapping of the parameter into the internal state used in MCMC calculations, and (for differential evolution simulations) a `random` element that defines the distribution to be used for the random perturbation to be added to this parameter in proposals.

The `name` element can specify subparameters, elements in an array of parameters, and elements within a parameter's `value` element. For example, consider the parameter file:

```
<parameter1 value="123"/>
<parameterA value="456"/>
<parameterA value="789"/>
<parameterA value="987"/>
<parameter2 value="abc">
 <parameter2a value="def"/>
</parameter2>
<parameterX value="1.0 2.0 3.0"/>
```

- A name of `parameter1` will change the value (`123`) of the `parameter1` element;
- A name of `parameterA[1]` will change the value (`1789`) of the second `parameterA` element (array indexing is 0-offset);
- A name of `parameter2->parameter2a` will change the value (`def`) of the `parameter2a` element;
- A name of `parameterX1` will change the value (`2.0`) of second entry in the value of the `parameterX` element.

Currently allowed mappings are:

`linear` Effectively a null mapping, as the parameter is mapped into itself: $x \rightarrow x$.

`logarithmic` The parameter is mapped logarithmically: $x \rightarrow \log(x)$. This mapping can be applied only to uniform priors.

Note that the mapping will map the values of the prior. For example, if you specify a uniform prior with a logarithmic mapping, the upper and lower limits of the prior should be specified on x, not on $\log(x)$. These limits will be mapped appropriately internally.

Loading External Parameters/Priors

It is also possible to load parameters and their priors from external files. This is useful to add common sets of parameters, such as cosmological parameter. To do so, add an element of the form:

```
<xi:include href="../../constraints/parameters/wmap9Cosmology.xml"
   xmlns:xi="http://www.w3.org/2001/XInclude" />
```

after the `parameters` section of the constraint file. The `href` attribute must give the path (relative to the constraint file, or absolute) to the external parameter file. This file should contain its own `parameters` block, describing all parameters to be varied along with their priors.

Derived Parameter Values

It is possible to define parameters in terms of other parameters. Common uses for this include:

- Setting Ω_Λ from the value of Ω_M to enforce a flat Universe;
- Setting the values of parameters with correlated priors as linear combinations of dummy parameters for which the priors are independent.

To define a parameter in this way include a `parameter` element of the form:

```
<parameter>
 <name>sigma_8</name>
 <define>0.8178+%cosmology0*0.003817+%cosmology1*0.007931+%cosmology2*0.01002
    +%cosmology3*0.001584+%cosmology4*0.002931+%cosmology5*0.001727</define>
</parameter>
```

Here the `define` element gives an equation for the parameter in terms of other parameters. All standard mathematical operators and functions (as recognized by Perl) can be used, and other parameters referenced by using their name prefixed with a "%".

Including External Parameters

Predefined sets of parameters (along with their priors) can be included using the `xi:include` element. For example,

```
 <xi:include href="../../constraints/parameters/wmap7Cosmology.xml"
    xmlns:xi="http://www.w3.org/2001/XInclude" />
```

will include a set of parameters from the file `../../constraints/parameters/wmap7Cosmology.xml` which defines priors on cosmological parameters consistent with the covariance matrix of the WMAP-7 cosmological constraints [Komatsu et al., 2010].

9.7.11. Distributions

Various distribution functions (for priors and random perturbations) are supported. Where needed, the type of distribution should be specified in a `type` element. Additional parameters of the distribution are specified using the elements described in the following subsections.

`uniform`

A uniform distribution over a finite range

$$P(x) \propto \begin{cases} 1 & \text{if } x_\mathrm{l} \leq x \leq x_\mathrm{u} \\ 0 & \text{otherwise.} \end{cases} \tag{9.71}$$

Specified using:

`minimum` The lower limit of the range, x_l;

`maximum` The upper limit of the range, x_u.

`logUniform`

A distribution uniform in the logarithm of x over a finite range

$$P(x) \propto \begin{cases} x^{-1} & \text{if } x_\mathrm{l} \leq x \leq x_\mathrm{u} \\ 0 & \text{otherwise.} \end{cases} \tag{9.72}$$

Specified using:

`minimum` The lower limit of the range, x_l;

`maximum` The upper limit of the range, x_u.

`normal`

A normal distribution, optionally with lower and upper limits:

$$P(x) \propto \begin{cases} \exp[-(x-\mu)^2/2S] & \text{if } x_\mathrm{l} \leq x \leq x_\mathrm{u} \\ 0 & \text{otherwise.} \end{cases} \tag{9.73}$$

Specified using:

`mean` The mean, μ;

`variance` The variance, S;

`minimum` The lower limit of the range, x_l;

`maximum` The upper limit of the range, x_u.

`Cauchy`

A Cauchy distribution:

$$P(x) \propto \left[1 + \frac{x - x_0}{\gamma}\right]^{-1}. \tag{9.74}$$

Specified using:

`median` The median, x_0;

`scale` The scale, γ;

`StudentT`

Student's t-distribution:

$$P(x) \propto \left(1 + \frac{x^2}{\nu}\right)^{-(\nu+1)/2} \tag{9.75}$$

Specified using:

`degreesOfFreedom` The number of degrees of freedom, ν.

Part III.

Physical Implementation

In this Part we describe the physical implementation of galaxy formation in Galacticus, including all components and their properties and additional output quantities from the code.

10. Definitions and Conventions Used in Galacticus

Galacticus adopts various definitions and conventions internally. These are explained below.

10.1. Halo Masses and Dark Matter Mass

Halo masses require some care in specifying exactly what mass they represent due to the way in which merger trees are typically created. For example, when merger trees are extracted from N-body simulations, those simulations frequently represent *all* matter as collisionless. That is, the simulation contains a density $\Omega_\mathrm{M} = \Omega_\mathrm{DM} + \Omega_\mathrm{b}$ which is the sum of dark and baryonic matter densities, but all of this mass is represented as collisionless particles. Similarly, masses in merger trees built through Monte Carlo techniques typically represent all mass as collisionless.

The exact way in which masses within Galacticus are defined and used in specified in the following subsections.

10.1.1. Masses in the Basic Component

The `basic` component (see §11.8) tracks the mass of each halo as defined in the merger tree. As such, it should be considered to be the mass which the halo would have if baryonic matter behaved just as dark matter. Note that these masses are inclusive of subhalos—that is, the mass of a host halo includes the mass of all of its subhalos.

10.1.2. Dark Matter Profiles

The dark matter profile functions (see §12.11.2) return masses and densities etc. which are normalized to match the mass of the `basic` component at the virial radius of the halo. As such, their returned values should be considered to represent the case where baryonic matter behaves as dark matter. This is a convention, and is useful for calculations of large scale structure for example.

10.1.3. Galactic Structure Functions

The various galactic structure functions assume that the masses/densities/etc. reported by the dark matter profile functions should be scaled by a factor $(\Omega_\mathrm{M} - \Omega_\mathrm{b})/\Omega_\mathrm{b}$ to leave only the dark matter part of the profile. Baryonic contributions to the mass/density/etc. will be provided by the components representing those mass distributions.

10.1.4. Satellite Virial Orbits

These functions (see §12.51.2) typically use the `basic` component mass in determining parameters of an orbit, since they are typically calibrated to simulations of collisionless matter only.

10.1.5. Satellite Merging Timescales

These functions (see §12.51.1) typically use the `basic` component mass in determining parameters of an orbit, since they are typically calibrated to simulations of collisionless matter only.

10.1.6. Dynamical Friction

These functions (see §12.41) evaluate densities through the relevant galactic structure function, and so correctly account for the fraction of the `basic` component mass which is in the form of dark matter.

10.1.7. Galactic Structure Radius Solvers

These functions §12.16) determine the radii of galactic components (such as disk and spheroid), typically by iteratively seeking a solution in which their angular momenta and radii are consistent (assuming rotational support) with the net gravitational potential of the entire system (galaxy plus dark matter halo).

10.2. Luminosity Units

Galaxy luminosities are output in the AB magnitude system, such that a luminosity of 1 corresponds to an object of $0^{\rm th}$ absolute magnitude in the AB magnitude system. This implies that the luminosities are in units of 4.4659×10^{13} W/Hz.

10.3. Peculiar Velocities

Velocities in Galacticus are always *physical* velocities. When reading merger tree properties (including velocities) from file it is often convenient to store velocities without the Hubble flow contribution, as "peculiar velocities", in the file—see §A.7 for how to specify whether or not the velocities included in the file include the Hubble flow or not.

If peculiar velocities are stored it is important to use the same definition of pecular velocity as is used by Galacticus. Defining t to be physical time and $\mathbf{x}$ to be comoving position, Galacticus uses the conventional definition of peculiar velocity in a cosmological context, namely that it is the deviation of the physical velocity from the Hubble flow. Physical coordinates are given by $\mathbf{r} = a\mathbf{x}$, so the peculiar velocity is

$$\mathbf{v}_{\rm pec} \equiv \frac{{\rm d}\mathbf{r}}{{\rm d}t} - H\mathbf{r} = a\frac{{\rm d}\mathbf{x}}{{\rm d}t} = \frac{{\rm d}\mathbf{x}}{{\rm d}\eta}, \tag{10.1}$$

where ${\rm d}\eta = {\rm d}t/a$ is conformal time.

10.4. Gravitational Potentials

Gravitational potentials are measured in velocity units (i.e. $\mathrm{km}^2/\mathrm{s}^2$), and the arbitrary constant offset is chosen such that the total gravitational potential in any halo at the virial radius is $\Phi(r_{\rm virial}) = -V^2_{\rm virial}$. This choice is made for two reasons:

1. some mass distributions used have potentials which diverge as $r \rightarrow \infty$, so the usual choice of $\Phi(r) \rightarrow 0$ as $r \rightarrow \infty$ is not applicable;

2. this choice is consistent with the potential at the virial radius of the halo considered as a point mass as is used in Keplerian orbit calculations.

Note that the choice of constant offset for the potential of any mass distribution or galactic component is irrelevant—the galactic structure function which computes potential will ensure that the potential is always offset to match the definition given above.

11. Node Components

In addition to the implementations described here, each component class has a "`null`" implementation. Selecting this implementation—which has no properties and does not respond to any events—effectively switches off the relevant component class. Of course, this is safe only if none of the other active implementations expect to get or set properties of the component class (or if they rely on a sensible implementation of that class).

11.1. (Supermassive) Black Hole

11.1.1. "Standard" Implementation

Properties

The standard black hole implementation defines the following properties:

`mass` The mass of the black hole: $M_\bullet$ `[blackHoleMass]`.

`spin` The spin of the black hole, $j_\bullet$ `[blackHoleSpin]`.

`radialPosition` The radial position of the black hole: $r_\bullet$ `[blackHoleRadialPosition]`

Initialization

Black holes are not initialized, they are created (with a seed mass given by `blackHoleSeedMass` and zero spin) as needed.

Differential Evolution

In the standard black implementation the mass and spin evolve as:

$$\dot{M}_\bullet = (1-\epsilon_{\rm radiation}-\epsilon_{\rm jet})\dot{M}_0 \quad (11.1)$$

$$\dot{j}_\bullet = \dot{j}(M_\bullet, j_\bullet, \dot{M}_0), \quad (11.2)$$

where $\dot{M}_0$ is the rest mass accretion rate, $\epsilon_{\rm radiation}$ is the radiative efficiency of the accretion flow feeding the black hole, $\epsilon_{\rm jet}$ is the efficiency with which accretion power is converted to jet power and $\dot{j}(M_\bullet, j_\bullet, \dot{M}_0)$ is the spin-up function of that accretion flow (see §16.4.1). The rest mass accretion rate is computed assuming Bondi-Hoyle-Lyttleton accretion from the spheroid gas reservoir (with an assumed temperature of `[bondiHoyleAccretionTemperatureSpheroid]`) enhanced by a factor of `[bondiHoyleAccretionEnhancementSpheroid]` and from the host halo (with whatever temperature that hot halo temperature profile specifies; see §12.27) enhanced by a factor of `[bondiHoyleAccretionEnhancementHotHalo]`. For accretion from the hot halo, the Bondi radius is limited to the outer radius of the hot halo. Additionally, the accretion rate is limited to:

$$\dot{M}_{0,\ \text{hot halo, maximum}} = M_{\rm hot}/\tau_{\text{sound crossing}}, \quad (11.3)$$

where $\tau_{\text{sound crossing}} = r_{\text{hot halo outer}}/c_{\rm s}$ where $r_{\text{hot halo outer}}$ is the outer radius of the hot halo and $c_{\rm s}$ is the speed of sound in the hot halo.

If `[bondiHoyleAccretionHotModeOnly]=true` then the accretion occurs only from that fraction of the hot halo gas which was accreted in the "hot mode", otherwise accretion occurs from the entirety of the hot halo reservoir. In the first case a simple estimate of the hot mode fraction is made:

$$f_\mathrm{hot} = \begin{cases} 1 & \text{if } x < 0.9 \\ y(x)^2[2y(x)-3]+1 & \text{if } 0.9 \leq x \leq 1.0 \\ 0 & \text{if } x > 1.0, \end{cases} \tag{11.4}$$

where $x = r_\mathrm{cool}/r_\mathrm{virial}$ and $y(x) = [x-0.9]/[1.0-0.9]$.

The rest mass accretion rate is removed (as a mass sink) from the spheroid and hot halo components appropriately. The black hole is assumed to cause feedback in two ways:

Radio-mode If `[blackHoleHeatsHotHalo]=true` then any jet power from the black hole-accretion disk system (see §12.5) is included in the hot halo heating rate providing that the halo is in the slow cooling regime[1] (i.e. if the cooling radius is smaller than the virial radius; see, for example, Benson and Bower 2010);

Quasar-mode A mechanical wind luminosity of [Ostriker et al., 2010]

$$L_\mathrm{wind} = \epsilon_{\bullet,\mathrm{wind}} H(\epsilon_\mathrm{radiation}, 1, s) \dot{M}_0 \mathrm{c}^2, \tag{11.5}$$

where $\epsilon_{\bullet\mathrm{wind}}$ =`[blackHoleWindEfficiency]` is the black hole wind efficiency, s=`blackHoleWindEfficiencyScalesWithRadiativeEfficiency`, and

$$H(a,b,c) = \begin{cases} a & \text{if } c = \texttt{true} \\ b & \text{if } c = \texttt{false}, \end{cases} \tag{11.6}$$

is added to the gas component of the spheroid (which, presumably, will respond with an outflow for example—see §11.5 for details of how specific implementations of the spheroid component respond to the addition of energy) if and only if the wind pressure (at the spheroid characteristic radius) is less than the typical thermal pressure in the spheroid gas [Ciotti et al., 2009], i.e.

$$\begin{aligned} P_\mathrm{wind} &< P_\mathrm{ISM} \\ \frac{1}{2}\rho_\mathrm{wind} V_\mathrm{wind}^2 &< \frac{3 \mathrm{k_B} T_\mathrm{ISM} \langle \rho_\mathrm{ISM} \rangle}{2 m_\mathrm{H}}. \end{aligned} \tag{11.7}$$

Since $\Omega r^2 \rho_\mathrm{wind} V_\mathrm{wind}^3 = L_\mathrm{wind}$ where Ω is the solid angle of the wind flow, this can be rearranged to give $\langle \rho_\mathrm{ISM} \rangle > \rho_\mathrm{wind,critical}$ where

$$\rho_\mathrm{wind,critical} = \frac{2 m_\mathrm{H} L_\mathrm{wind}}{3 \Omega r^2 V_\mathrm{wind} \mathrm{k_B} T_\mathrm{ISM}}. \tag{11.8}$$

This critical wind density is computed at the characteristic radius of the spheroid, r_spheroid, assuming $V_\mathrm{wind} = 10^4 \mathrm{km/s}$, $T_\mathrm{ISM} = 10^4 \mathrm{K}$ and $\Omega = \pi$, and the ISM density is approximated by

$$\langle \rho_\mathrm{ISM} \rangle = \frac{3 M_\mathrm{gas,spheroid}}{4\pi} r_\mathrm{spheroid}^3. \tag{11.9}$$

For numerical ease, the fraction, f_wind, of the wind luminosity added to the spheroid is adjusted smoothly through the $\rho_\mathrm{ISM} \approx \rho_\mathrm{wind,critical}$ region according to

$$f_\mathrm{wind} = \begin{cases} 0 & \text{if } x < 0, \\ 3x^2 - 2x^3 & \text{if } 0 \leq x \leq 1, \\ 1 & \text{if } x > 1, \end{cases} \tag{11.10}$$

where $x = \rho_\mathrm{ISM}/\rho_\mathrm{wind,critical} - 1/2$.

[1] Specifically, the jet power multiplied by $f_\mathrm{hot}[(M_\mathrm{hot}/M_\mathrm{total})(\Omega_\mathrm{M}/\Omega_\mathrm{b})]^2$ is added to the hot halo heating rate. The dependence on the gas fraction in the hot halo ensures that the heating rate goes smoothly to zero as the hot halo becomes depleted of gas.

The radial position, $r_\bullet$, evolves according to the selected radial migration method (see §16.4.1).

Interactions between black hole triplets are accounted for if `[tripleBlackHoleInteraction]=true` (and if at least three black holes exist within the node of course). In this case the triple is treated as consisting of an inner binary (assumed to be the central black hole and the black hole closest to it) and a third, singleton black hole. When the tertiary black hole reaches a separation of

$$a_\mathrm{h} = \frac{\mathrm{G}(M_{\bullet,1} + M_\mathrm{bullet,2})}{4\sigma^2} \tag{11.11}$$

it is assumed to undergo a triple interaction with the binary. Once a triple interaction occurs, no further triple interaction for the specific tertiary black hole can occur unless the host galaxy merges with another galaxy, at which point the black holes from the merging galaxy are eligible for another triple interaction in their new host.

The logic of what happens in a triple black hole interaction is taken from Volonteri et al. [2003]. Labelling the central black hole as 1, its binary partner as 2 and the tertiary black hole as 3, and defining

$$q_3 = \frac{M_{\bullet,3}}{M_{\bullet,1} + M_{\bullet,2}}, \tag{11.12}$$

then if $q_3 \leq 2$ then, if $M_{\bullet,3} \leq M_{\bullet,2}$ we set

$$a_3 = \frac{a_2}{1 + 0.4q_3}, \tag{11.13}$$

and define

$$E_\mathrm{bind} = \frac{\mathrm{G}M_{\bullet,3}M_{\bullet,1}}{a_3}, \tag{11.14}$$

and

$$\Delta K = 0.4q_3 E_\mathrm{bind}, \tag{11.15}$$

$i = 3$ and $j = 2$.

Otherwise if $q_3 \leq 2$ and $M_{\bullet,3} > M_{\bullet,2}$ we set

$$a_3 = \frac{a_3}{1 + 0.4q_3}, \tag{11.16}$$

and define

$$E_\mathrm{bind} = \frac{\mathrm{G}M_{\bullet,2}M_{\bullet,1}}{a_2}, \tag{11.17}$$

and

$$\Delta K = 0.4q_3 E_\mathrm{bind}, \tag{11.18}$$

$i = 2$ and $j = 3$.

Finally, if $q_3 > 2$, then we set

$$a_3 = 0.53a_3, \tag{11.19}$$

and define

$$E_\mathrm{bind} = \frac{\mathrm{G}M_{\bullet,2}M_{\bullet,1}}{a_2}, \tag{11.20}$$

and

$$\Delta K = 0.9q_3 E_\mathrm{bind}, \tag{11.21}$$

$i = 2$, $j = 3$.

Black hole i is identified as the "ejected" hole, with black hole j becoming the new binary member. Therefore

$$M_{\bullet,ejected} = M_{\bullet,i}. \tag{11.22}$$

and

$$M_{\rm binary} = M_{\bullet,\rm j} + M_{\bullet,1}. \tag{11.23}$$

The imparted velocities of these two systems are

$$V_{\rm ejected} = \left[\frac{2\Delta K}{(1 + M_{\bullet,ejected}/M_{\rm binary}) M_{\bullet,ejected}} \right]^{1/2} \tag{11.24}$$

and

$$V_{\rm binary} = \left[\frac{2\Delta K}{(1 + M_{\rm binary}/M_{\bullet,ejected}) M_{\rm binary}} \right]^{1/2}. \tag{11.25}$$

If

$$\frac{1}{2} V^2_{\rm ejected|binary} + \Phi(a_{\rm ejected|binary}) \geq 0 \tag{11.26}$$

for either velocity, then that system is ejected from the node. Ejected black holes are removed from the node. If the binary is ejected the central black hole is replaced with a "null", zero mass placeholder.

Event Evolution

Node mergers: None.

Satellite merging: The black holes in the two merging galaxies can be instantaneously merged, or taken at an initial separation (see §12.54), it is then evolved until reaching zero separation whereupon it is assumed to undergo merger. Properties are computed using the selected black hole binary merger method (see §16.4.1). In addition, the recoil velocity of the new black hole due to gravitational wave emission is computed using the selected method (see §12.56), and if greater than the potential at the center of the galaxy, is assumed to have escaped the galaxy. Black holes which escape the galaxy are simply discarded and no longer tracked. For computational purposes, they are replaced with a "null", zero mass black hole at the center of the galaxy. If any other black hole comes within a distance

$$a_{\rm h} = \frac{{\rm G}M_\bullet}{4\sigma^2}, \tag{11.27}$$

where σ is approximated to be the virial velocity of the dark matter halo, it is promoted to being the new "central" black hole of the node.

Node promotion: None.

Additional Output

If the `[blackHoleOutputAccretion]` input parameter is set to true, then rest mass accretion rate (in $M_\odot$ Gyr^{-1}), jet power (in $M_\odot$ km^2 s^{-1} Gyr^{-1}) and radiative efficiency of the black hole[2] are output as `blackHoleAccretionRate`, `blackHoleJetPower` and `blackHoleRadiativeEfficiency` respectively.

If the `[blackHoleOutputData]` input parameter is set to true, then the Masses (in $M_\odot$), Spins (for now just a scalar with no direction), final Radius (in Mpc), timescales (in Gyr) until merger, accretion rates (in $M_\odot perGyr$) and radiative Efficiencies of all the black holes in the galaxy are given as outputs in the `blackHole` section of the output hdf5. This also saves the tree node and merger tree index for further use when using the data.

The outputs of mergers are also automatically saved, as outputs in the `blackHoleMergers` section of the output hdf5. Those outputs are the time at which mergers happened and the mass ratio between the two merging black holes.

[2]Technically of the black hole plus accretion disk system.

11.1.2. "Simple" Implementation

Properties

The simple black hole implementation defines the following property:

`mass` The mass of the black hole: $M_\bullet$ `[blackHoleMass]`.

Initialization

Black holes are not initialized, they are created (with a seed mass given by `blackHoleSeedMass`) as needed.

Differential Evolution

In the simple black hole implementation the mass evolves as:

$$\dot{M}_\bullet = (1-\epsilon_{\rm wind})\epsilon_{\rm BH}\dot{M}_{\star,\rm spheroid} \tag{11.28}$$

$$\tag{11.29}$$

where $\epsilon_{\rm BH}$ is the ratio of rates at which the black hole and stellar spheroid grow. The black hole is assumed to cause feedback in two ways:

Radio-mode If `[blackHoleHeatsHotHalo]`=`true` and `[blackHoleAccretesFromHotHalo]`=`false` then a power $\epsilon_{\rm heat}\epsilon_{\rm BH}\dot{M}_{\star,\rm spheroid}{\rm c}^2$ where $\epsilon_{\rm heat}$ =`[blackHoleHeatingEfficiency]` is included in the hot halo heating rate providing that the halo is in the slow cooling regime (i.e. if the cooling radius is smaller than the virial radius; see, for example, Benson and Bower 2010) and the accretion rate onto the black hole is reduced by $\epsilon_{\rm heat}\epsilon_{\rm BH}\dot{M}_{\star,\rm spheroid}$. If `[blackHoleHeatsHotHalo]`=`true` and `[blackHoleAccretesFromHotHalo]`=`true` then a power $\epsilon_{\rm heat}\dot{M}_{\rm Eddington}{\rm c}^2$ is included in the hot halo heating rate providing that the halo is in the slow cooling regime and the accretion rate onto the black hole is increased[3] by $\dot{M}_{\rm Eddington}\epsilon_{\rm heat}(1-\epsilon_{\rm jet})/\epsilon_{\rm jet}$, where $\epsilon_{\rm jet}$ =`[blackHoleJetEfficiency]`;

Quasar-mode A mechanical wind luminosity of [Ostriker et al., 2010]

$$L_{\rm wind} = \epsilon_{\bullet,wind}\dot{M}_0{\rm c}^2, \tag{11.30}$$

where $\epsilon_{\bullet wind}$ =`[blackHoleWindEfficiency]` is the black hole wind efficiency, is added to the gas component of the spheroid (which, presumably, will respond with an outflow for example).

Event Evolution

Node mergers: None.

Satellite merging: The black holes in the two merging galaxies are instantaneously merged. Properties are computed using the selected black hole binary merger method (see §16.4.1).

Node promotion: None.

Additional Output

If the `[blackHoleOutputAccretion]` input parameter is set to true, then rest mass accretion rate (in $M_\odot$ Gyr^{-1}) is output as `blackHoleAccretionRate`.

[3] Note that mass is not removed from the hot halo to compensate, since the accretion rate is independent of the hot halo mass this could lead to negative mass in the halo.

11.2. Dynamics Statistics

This class collects statistics related to galaxy dynamics.

11.2.1. "Bars" Implementation

Properties

The "bars" dynamics statistics implementation defines the following properties:

`time` An array of times at which bar statistics are recorded.

`barInstabilityTimescale` The timescale of bar instability at each tabulated time.

`adiabaticRatio` The adiabatic ratio, a, of the galaxy at each tabulated time defined as:

$$a = \left(\frac{r_{\rm peri}}{v_{\rm peri}} \right) \left(\frac{2\pi r_{\rm disk}}{v_{\rm disk}} \right)^{-1}, \tag{11.31}$$

where $r_{\rm peri}$ is the pericentric distance of the galaxies orbit, $v_{\rm peri}$ is the orbital velocity at pericenter, $r_{\rm disk}$ is the characteristic radius of the disk, and $v_{\rm disk}$ is the circular velocity at that radius. For non-satellite galaxies the adiabatic ratio is set to -1.

Bar statistics are recorded at a frequency given by `[dynamicsStatisticsBarsFrequency]` times the host halo dynamical time.

Initialization

Component is not initialized, it is created as soon as a disk exists.

Differential Evolution

N/A.

Event Evolution

Node mergers: None.

Satellite merging: None.

Node promotion: None.

Additional Output

Bar statistics time series are output for each node to a group `Outputs/Output<O>/dynamicsStatistics/tree<T>` (where `<O>` is the output number and `<T>` is the tree index) into datasets named `time<N>`, `timeScale<N>`, and `adiabaticRatio<N>` (where `<N>` is the node index).

11.3. Hot Halo

11.3.1. "Very Simple" Implementation

Properties

The very simple hot halo implementation defines the following properties:

`mass` The mass of gas in the hot halo: $M_{\rm hot}$ `[hotHaloMass]`.

and the following pipes:

`hotHaloCoolingMass` The net cooling rate of gas mass is sent through this pipe. Any component may claim this pipe and connect to it, allowing it to receive the cooling gas.

`outflowingMass` Galactic components that wish to expel gas due to an outflow can send that mass through this pipe, where it will be received into the hot halo component.

Initialization

At initialization, nodes are assigned a mass of gas equal to their own mass, minus the mass of any progenitors, multiplied by $\Omega_{\rm b}/\Omega_{\rm Matter}$.

Differential Evolution

In the very simple hot halo implementation the hot gas mass and heavy element mass(es) evolves as:

$$\dot{M}_{\rm hot} = -\dot{M}_{\rm cooling} + \dot{M}_{\rm outflow}, \tag{11.32}$$

where $\dot{M}_{\rm cooling}$ is the rate of mass loss from the hot halo due to cooling (see §12.9.2. In the above $\dot{M}_{\rm outflow}$ is the net rate of outflow from any components in the node. For satellite galaxies, the outflow is instead directed to the hot halo of the host node.

Event Evolution

Node mergers: Any hot gas from the merging halo is transferred to its host halo.

Satellite merging: Any hot halo of the satellite node is added to that of the host node and the hot halo component removed from the satellite node.

Node promotion: Any hot halo of the parent node is added to that of the node prior to promotion.

Halo formation: None.

11.3.2. "Very Simple Delayed" Implementation

Properties

The delayed very simple hot halo implementation extends the "very simple" implementation by adding a reservoir for outflowed gas from which the hot halo is gradually replenished. It defines the following properties:

`outflowedMass` The mass of gas in the outflowed reservoir of the hot halo: $M_{\rm outflowed}$ `[hotHaloOutflowedMass]`.

Initialization

At initialization, nodes are assigned zero outflowed mass.

Differential Evolution

This implementation steals the `outflowingMass` pipe from the very simple component and redirects it to the outflowed mass reservoir. The resulting rates of change of hot and outflowed masses are then:

$$\begin{aligned} \dot{M}_\mathrm{hot} &= -\dot{M}_\mathrm{cooling} + \dot{M}_\mathrm{reincorporation}, \\ \dot{M}_\mathrm{outflowed} &= +\dot{M}_\mathrm{outflow} - \dot{M}_\mathrm{reincorporation}, \end{aligned} \tag{11.33}$$

where $\dot{M}_\mathrm{reincorporation}$ is the reincorporation rate of outflowed gas (see §12.26).

Event Evolution

Node mergers: Any outflowed gas from the merging halo is transferred to its host halo.

Satellite merging: Any outflowed halo of the satellite node is added to that of the host node and the outflowed mass component removed from the satellite node.

Node promotion: Any outflowed gas of the parent node is added to that of the node prior to promotion.

Halo formation: None.

11.3.3. "Standard" Implementation

Properties

The standard hot halo implementation defines the following properties:

`unaccretedMass` The mass of gas which could have accreted onto the halo if it always accreted baryons and dark matter in the universal proportion, but which failed to do so (e.g. perhaps due to being photoheated to a high temperature and so being resistant to accretion into shallow potential wells): M_failed.

`mass` The mass of gas in the hot halo: M_hot `[hotHaloMass]`.

`angularMomentum` The angular momentum of the gas in the hot halo, J_hot `[hotHaloAngularMomentum]`.

`abundances` The mass(es) of heavy elements in gas in the hot halo, $M_{Z,\mathrm{hot}}$ `[hotHalo{abundanceName}]`.

`outflowedMass` The mass of gas from outflows in the hot halo: $M_\mathrm{outflowed}$ `[hotHaloOutflowedMass]`.

`outflowedAngularMomentum` The angular momentum of the outflowed gas in the hot halo, $J_\mathrm{outflowed}$ `[hotHaloOutflowedAngularMomentum]`.

`outflowedAbundances` The mass(es) of heavy elements in outflowed gas, $M_{Z,\mathrm{outflowed}}$ `[hotHaloOutflowed{abundanceName}]`

`chemicals` The mass(es) of molceules in the hot gas, M_chemical `[hotHaloChemicals{chemicalName}]`.

`outerRadius` The outer boundary radius fo the hot halo: $r_\mathrm{hot,outer}$ `[hotHaloOuterRadius]`.

`strippedMass` The mass of gas which has been stripped from the hot halo (by ram pressure or tidal forces for example): $M_{\rm hot,stripped}$. This property is computed only if `[hotHaloTrackStrippedGas]=true`.

`strippedAbundances` The mass(es) of heavy elements in gas that has been stripped from the hot halo (by ram pressure or tidal forces for example), $M_{Z,\rm hot,stripped}$. These properties are computed only if `[hotHaloTrackStrippedGas]=true`.

and the following pipes:

`heatSource` Energy sent through this pipe is added to the hot halo and used to offset the cooling rate (see below; heat pushed should be in units if $M_\odot$ (km/s)^2 Gyr^{-1}).

`cooling[Mass|Angular_Momentum|Abundances]_To` The net cooling rate of gas mass (and metal content and magnitude of angular momentum) is sent through this pipe. Any component may claim this pipe and connect to it, allowing it to receive the cooling gas.

`outflowing[Mass|AngularMomentum|Abundances]_To` Galactic components that wish to expel gas due to an outflow can send that mass (plus metals and angular momentum) through this pipe, where it will be received into the hot halo component.

`massSink` Removes gas (and proportionate amounts of angular momentum and elements) from the hot gas halo.

Initialization

At initialization, any nodes with no children are assigned a hot halo mass, and failed accreted mass as dictated by the baryonic accretion method (see §12.2) and angular momentum based on the accreted mass and the halo spin parameter.

Differential Evolution

In the standard hot halo implementation the hot gas mass and heavy element mass(es) evolves as:

$$\dot{M}_{\rm failed} = \dot{M}_{\rm failed\ accretion} \tag{11.34}$$

$$\dot{M}_{\rm hot} = \dot{M}_{\rm accretion} - \dot{M}_{\rm cooling} + \dot{M}_{\rm outflow,return} - \dot{M}_{\rm expelled} - \dot{M}_{\rm hot,stripped}, \tag{11.35}$$

$$\dot{M}_{Z,\rm hot} = -\dot{M}_{\rm cooling}\frac{M_{Z,\rm hot}}{M_{\rm hot}} + \dot{M}_{Z,\rm outflow,return} - \dot{M}_{Z,\rm expelled} - \dot{M}_{Z,\rm hot,stripped}, \tag{11.36}$$

$$\dot{M}_{\rm chemical} = -[\dot{M}_{\rm cooling} + \dot{M}_{\rm expelled} + \dot{M}_{\rm hot,stripped}]\frac{M_{\rm chemical}}{M_{\rm hot}} + f_{\rm chemical,outflow}\dot{M}_{\rm outflow,return} + \dot{M}_{\rm chemical,reactions}, \tag{11.37}$$

$$\dot{r}_{\rm hot,outer} = \begin{cases} \frac{r_{\rm rp}-r_{\rm hot,outer}}{\tau_{\rm dynamical}} & \text{if } r_{\rm rp} < r_{\rm hot,outer} \\ 0 & \text{otherwise.} \end{cases} \tag{11.38}$$

$$\dot{M}_{\rm hot,stripped} = -4\pi\rho_{\rm hot}(r_{\rm hot,outer})r_{\rm hot,outer}^2\dot{r}_{\rm hot,outer} + \dot{M}_{\rm outflows}f_{\rm outflow,stripped} \tag{11.39}$$

$$\dot{M}_{Z,\rm hot,stripped} = -4\pi\rho_{\rm hot}(r_{\rm hot,outer})r_{\rm hot,outer}^2\dot{r}_{\rm hot,outer}(M_{Z,\rm hot}/M_{\rm hot}) + \dot{M}_{Z,\rm outflows}f_{\rm outflow,stripped} \tag{11.40}$$

where $r_{\rm rp}$ is the ram pressure stripping radius as computed by the `hotHaloRamPressureStrippingMethod` method (see §12.24), $\dot{M}_{\rm accretion}$ is the rate of growth of the hot component due to accretion from the IGM and $\dot{M}_{\rm failed\ accretion}$ is the rate of failed accretion from the IGM (these may include a component due to transfer of mass from the failed to accreted reservoirs) and $\dot{M}_{\rm cooling}$ is the rate of mass

loss from the hot halo due to cooling (see §12.9.2—cooling rates are computed using the current node if `[hotHaloCoolingFromNode]=current node` or from the formation node if that parameter is set to `formation node`) minus any heating rate defined as

$$\dot{M}_{\rm heating} = \dot{E}_{\rm input}/V^2_{\rm virial}, \tag{11.41}$$

where $\dot{E}_{\rm input}$ is the rate at which energy is being sent through the "energy input" pipe and $V_{\rm virial}$ is the virial velocity of the halo. The net cooling rate is never allowed to drop below zero. If the mass heating rate exceeds the mass cooling rate and `[hotHaloExcessHeatDrivesOutflow]=false` then the excess energy is not used and $\dot{M}_{\rm expelled}=0$. Alternatively, if `[hotHaloExcessHeatDrivesOutflow]=true` then

$$\dot{M}_{\rm expelled} = \begin{cases} \alpha_{\rm expel} M_{\rm hot}/\tau_{\rm dynamical} & \text{if } \dot{M}_{\rm heating} - \dot{M}_{\rm cool} > \alpha_{\rm expel} M_{\rm hot}/\tau_{\rm dynamical} \\ \dot{M}_{\rm heating} - \dot{M}_{\rm cool}, & \text{otherwise,} \end{cases} \tag{11.42}$$

where $\dot{M}_{\rm cool}$ is the intrinsic cooling rate in the halo (i.e. the cooling rate in the absence of any heating) and $\alpha_{\rm expel}=$`[hotHaloExpulsionRateMaximum]` limits the maximum rate at which mass can be expelled from the halo.

In the above, $f_{\rm chemical,return}$ if the mass fraction of each chemical species in the outflowed gas and is assumed to be equal to that given by the atomic ionization state functions (see §12.30) at the virial temperature and mean density of the halo. Finally, $\dot{M}_{\rm chemical,reactions}$ represents the rate of change of masses of chemical species due to chemical and atomic processes and is computed using the chemical rates functions (see §12.39). The angular momentum of the hot gas evolves as:

$$\dot{J}_{\rm hot} = \dot{M}_{\rm accretion}\frac{\dot{J}_{\rm node}}{\dot{M}_{\rm node}} - \dot{M}_{\rm cooling} r_{\rm cool} V_{\rm rotate} + \dot{J}_{\rm outflow,return} - \dot{M}_{\rm expelled}\frac{J_{\rm hot}}{M_{\rm hot}}, \tag{11.43}$$

where $\dot{M}_{\rm node}$ and $\dot{J}_{\rm node}$ are defined in §11.8. For the outflowed components:

$$\dot{M}_{\rm outflowed} = -\dot{M}_{\rm outflow,return} + \dot{M}_{\rm outflows}(1-f_{\rm outflow,stripped}), \tag{11.44}$$

$$\dot{M}_{Z,\rm outflowed} = -\dot{M}_{Z,\rm outflow,return} + \dot{M}_{Z,\rm outflows}(1-f_{\rm outflow,stripped}), \tag{11.45}$$

$$\tag{11.46}$$

and:

$$\dot{J}_{\rm outflowed} = -\dot{J}_{\rm outflow,return} + \dot{J}_{\rm outflows}. \tag{11.47}$$

In the above

$$\dot{M}|\dot{M}_Z|\dot{J}_{\rm outflow,return} = \alpha_{\rm outflow\ return\ rate}\frac{M|M_Z|J_{\rm outflowed}}{\tau_{\rm dynamical,halo}}, \tag{11.48}$$

where $\alpha_{\rm outflow\ return\ rate}=$ (`hotHaloOutflowReturnRate`) is an input parameter controlling the rate at which gas flows from the outflowed to hot reservoirs, and $\dot{M}|\dot{M}_Z|\dot{J}_{\rm outflows}$ are the net rates of outflow from any components in the node.

In the above, $f_{\rm outflow,stripped}$ is the fraction of outflowing material assumed to be stripped from the halo. The is computed following the algorithm of Font et al. [2008], namely

$$f_{\rm outflow,stripped} = \epsilon_{\rm strip}\frac{M_{\rm hot,outer}}{M_{\rm hot,virial}}, \tag{11.49}$$

where $\epsilon_{\rm strip}=$`[hotHaloOutflowStrippingEfficiency]` is an input parameter, $M_{\rm hot,outer}$ is the mass of hot gas contained within the outer radius of the hot halo and $M_{\rm hot,virial}$ is the mass of hot gas that would be present if the hot halo extended to the virial radius (i.e. if no stripping had occurred).

A fraction $1-$`[hotHaloAngularMomentumLossFraction]` of the cooling angular momentum rate, $\dot{M}_{\rm cooling} r_{\rm cool} V_{\rm rotate}$, is sent through the `Hot_Halo_Cooling_Angular_Momentum` pipe.

Event Evolution

Node mergers: If the `starveSatellites` parameter is true, then any hot halo properties of the minor node are added to those of the major node and the hot halo component removed from the minor node. Additionally in this case, any material outflowed or stripped from the the satellite galaxy to its hot halo is transferred to the hot halo of the host dark matter halo after each timestep. (Alternatively, if `starveSatellitesOutflowed=true` then only the outflowed and stripped gas is transferred to the host halo—the main hot gas reservoir is left in place.) If stripped mass is being tracked (i.e. if `[hotHaloTrackStrippedGas]=true`) then any stripped mass is transferred from the satellite galaxy to the hot halo of the host dark matter halo after each timestep. If `[hotHaloNodeMergerLimitBaryonFraction]=true` then the hot gas content of the merged node is limited such that the total baryon content of the node (including satellites) does not exceed the universal baryon fraction, if possible. Any gas removed to enforce this limit is placed into the unaccreted gas reservoir, from which is may eventually be reaccreted.

Satellite merging: If the `starveSatellites` parameter is false, then any hot halo properties of the satellite node are added to those of the host node and the hot halo component removed from the satellite node.

Node promotion: Any hot halo properties of the parent node are added to those of the node prior to promotion.

Halo formation: If `[hotHaloOutflowReturnOnFormation]=true` then all outflowed gas is returned to the hot gas reservoir on halo formation events (see §16.4.3).

11.3.4. "Outflow Tracking" Implementation

Properties

The outflow tracking hot halo implementation extends the "standard" implementation by adding the following properties:

`trackedOutflowMass` The mass of gas in the hot halo which arrived there directly via outflow: $M_{\rm outflow,track}$ `[hotHaloTrackedOutflowMass]`.

`trackedOutflowAbundances` The mass of elements in the hot halo which arrived there directly via outflow: $M_{Z,\rm outflow,track}$ `[hotHaloTrackedOutflowAbundances]`.

Initialization

Outflowed masses and element masses are initialized to zero.

Differential Evolution

The tracked outflow masses evolve according to:

$$\dot{M}|\dot{M}_{Z_{\rm outflow,track}} = \alpha_{\rm outflow\ return\ rate}\frac{M|M_{Z_{\rm outflowed}}}{\tau_{\rm dynamical,halo}} - M|M_{Z_{\rm outflow,track}}\dot{M}_{\rm expelled}/M \quad (11.50)$$

(11.51)

Event Evolution

Node mergers: None.

Satellite merging: None.

Node promotion: None.

Halo formation: None.

11.4. Galactic Disk

11.4.1. "Very Simple" Implementation

This implementation assumes a disk with no structural properties—it consists of just gas and stellar masses.

Properties

The very simple galactic disk implementation defines the following properties:

`massGas` The mass of gas in the disk: $M_{\rm disk,gas}$ [`diskMassGas`];

`massStellar` The mass of stars in the disk: $M_{\rm disk,stars}$ [`diskMassStellar`].

Initialization

No initialization is performed—disks are created as needed.

Differential Evolution

In the very simple galactic disk implementation the gas mass evolves as:

$$\dot{M}_{\rm disk,gas} = \dot{M}_{\rm cooling} - \dot{M}_{\rm outflow,disk} - \dot{M}_{\rm stars,disk}, \tag{11.52}$$

where the rate of change of stellar mass is

$$\dot{M}_{\rm stars,disk} = \Psi - \dot{R}, \tag{11.53}$$

where $\dot{R}$ is the rate of mass recycling from stars, and

$$\Psi = \frac{M_{\rm disk,gas}}{\tau_{\rm disk,star\ formation}} \tag{11.54}$$

with $\tau_{\rm disk,star\ formation}$ being the greater of the star formation timescale and $\Gamma_{\rm disk,starformation,minimum}\tau_{\rm dyn}$, where $\tau_{\rm dyn}$ is the dynamical time of the halo, and $\Gamma_{\rm disk,starformation,minimum}$ = `[diskStarFormationTimescaleMinimum]`. The outflow rate, $\dot{M}_{\rm outflow,disk}$, is computed for the current star formation rate and gas properties by the prescriptions for non-expulsive supernova feedback (see §12.52), but is limited to a maximum of $M_{\rm disk,gas}/\Gamma_{\rm disk,outflow,minimum}\tau_{\rm dyn}$, where $\Gamma_{\rm disk,outflow,minimum}$ = `[diskOutflowTimescaleMinimum]`. This outflow is piped to the hot halo component.

Event Evolution

Node mergers: None

Satellite merging: Disks may be destroyed (or, potentially, created or otherwise modified) as the result of a satellite merging event, as dictated by the selected merger remnant mass movement method (see §12.18.1).

Node promotion: None

11.4.2. "Very Simple Size" Implementation

This implementation extends the `verySimple` disk class by adding a half-mass radius property. No other assumptions about the mass distribution are made.

Properties

The very simple size galactic disk implementation defines the following additional properties:

`radius` The half-mass radius of the disk: $r_{\mathrm{disk},1/2}$ [`diskRadius`].

Initialization

No initialization is performed—disks are created as needed.

Differential Evolution

N/A—disk radii are computed using the selected galactic structure solver.

Event Evolution

Node mergers: None

Satellite merging: None

Node promotion: None

11.4.3. "Standard Implementation

This implementation assumes a disk with a cylindrical symmetry and a flattened profile in which stars trace gas. Currently, one option for the density profile is allowed[4]:

`exponentialDisk` Assumes an exponential profile, $\rho(R,z) = \rho_0 \exp(-r/r_{\mathrm{s}})\mathrm{sech}^2(z/z_{\mathrm{s}})$, for the disk component of a galaxy. The thickness, $z_{\mathrm{s}}/r_{\mathrm{s}}$ is set by the parameter [`heightToRadialScaleDisk`].

[4]The disk density distribution is handled internally using a `massDistribution` object (see §16.4.1). As such, any mass distribution implemented as an extension of the `massDistribution` class (and which is described by a single length scale) could be trivially addded to the standard disk component.

Properties

The standard galactic disk implementation defines the following properties:

`massGas` The mass of gas in the disk: $M_{\rm disk,gas}$ [`diskMassGas`].

`abundancesGas` The mass of elements in the gaseous disk: $M_{Z,\rm disk,gas}$ [`diskAbundancesGas{abundanceName}`].

`massStellar` The mass of stars in the disk: $M_{\rm disk,stars}$ [`diskMassStellar`].

`abundancesStellar` The mass of elements in the stellar disk: $M_{Z,\rm disk,stars}$ [`diskAbundancesStellar{abundanceName}`].

`luminositiesStellar` The luminosities (in multiple bands) of the stellar disk: $L_{\rm disk,stars}$ [`diskLuminositiesStellar{lumino`

`angularMomentum` The angular momentum of the disk, $J_{\rm disk}$ [`diskAngularMomentum`].

`radius` The radial scale length of the disk, $R_{\rm disk}$ [`diskRadius`].

`velocity` The circular velocity of the disk at $R_{\rm disk}$, $V_{\rm disk}$ [`diskVelocity`].

`massStellarFormed` The total mass of stars ever formed in the disk: $M_{\rm disk,stars,formed}$.

`fractionMassRetained` The fraction of mass retained in the disk as a result of transfer processes in which the transfer is proportional to the current mass: $f_{\rm retained}$.

Initialization

No initialization is performed—disks are created as needed.

Differential Evolution

In the standard galactic disk implementation the gas mass evolves as:

$$\dot{M}_{\rm disk,gas} = \dot{M}_{\rm cooling} - \dot{M}_{\rm outflow,disk} - \dot{M}_{\rm stars,disk} - \frac{M_{\rm disk,gas}}{\tau_{\rm bar}} - \dot{M}_{\rm rampressure} - \frac{M_{\rm disk,gas}}{M_{\rm disk,gas}+M_{\rm disk,stars}}\dot{M}_{\rm tidal}, \tag{11.55}$$

where the rate of change of stellar mass is

$$\dot{M}_{\rm stars,disk} = \Psi - \dot{R} - \frac{M_{\rm stars,disk}}{\tau_{\rm bar}} - \frac{M_{\rm disk,stars}}{M_{\rm disk,gas}+M_{\rm disk,stars}}\dot{M}_{\rm tidal}, \tag{11.56}$$

$$\dot{M}_{\rm stars,disk,formed} = \Psi, \tag{11.57}$$

with

$$\Psi = \frac{M_{\rm disk,gas}}{\tau_{\rm disk,star\ formation}} \tag{11.58}$$

with $\tau_{\rm disk,star\ formation}$ being the star formation timescale and $\dot{R}$ is the rate of mass recycling from stars and $\tau_{\rm bar}$ is a bar instability timescale (see §12.12). If `[diskStarFormationInSatellites]=true` then the star formation rate is forced to zero in satellite galaxies. The mass removed from the disk by the bar instability mechanism is added to the active spheroid component. Element abundances (including total metals) evolve according to:

$$\dot{M}_{Z,\rm disk,gas} = \dot{M}_{Z\rm cooling} - \dot{M}_{Z,\rm outflow,disk} - \dot{M}_{Z,\rm stars,disk} + \dot{y} - \frac{M_{Z,\rm disk,gas}}{M_{\rm disk,gas}}\dot{M}_{\rm rampressure} - \frac{M_{Z,\rm disk,gas}}{M_{\rm disk,gas}+M_{\rm disk,stars}}\dot{M}_{\rm tidal}, \tag{11.59}$$

and

$$\dot{M}_{Z,\mathrm{stars,disk}} = \Psi \frac{M_{Z,\mathrm{disk,gas}}}{M_{\mathrm{disk,gas}}} - \dot{R}_Z - \frac{M_{Z,\mathrm{disk,stars}}}{M_{\mathrm{disk,gas}} + M_{\mathrm{disk,stars}}} \dot{M}_{\mathrm{tidal}} \tag{11.60}$$

where $\dot{y}$ is the rate of element yield from stars and $\dot{R}_Z$ is the rate of element recycling. The angular momentum evolves as:

$$\dot{J}_{\mathrm{disk}} = \dot{J}_{\mathrm{cooling}} - \left[\dot{M}_{\mathrm{outflow,disk}} + \frac{M_{\mathrm{disk,gas}} + M_{\mathrm{disk,stars}}}{\tau_{\mathrm{bar}}} + \dot{M}_{\mathrm{rampressure}} + \dot{M}_{\mathrm{tidal}} \right] \frac{J_{\mathrm{disk}}}{M_{\mathrm{disk,gas}}}. \tag{11.61}$$

The outflow rate, $\dot{M}_{\mathrm{outflow,disk}}$, is computed for the current star formation rate and gas properties by the stellar properties subsystem (see §12.47) and prescriptions for expulsive and non-expulsive supernova feedback (see §12.53 and §12.52 respectively), but is not allowed to exceed $M_{\mathrm{gas,disk}}/\alpha_{\mathrm{outflowminimum,disk}}\tau_{\mathrm{disk,dynamical}}$, where $\tau_{\mathrm{disk,dynamical}} = R_{\mathrm{disk}}/V_{\mathrm{disk}}$ is the dynamical time of the disk and $\alpha_{\mathrm{outflowminimum,disk}} =$`[diskOutflowTimescaleMinimum` is the shortest timescale (in units of the dynamical timescale) on which gas can be removed from the disk. This limit prevents the disk being depleted on arbitrarily short timescales. The non-expulsive component of the outflow is piped to the hot halo component. The ram pressure and tidal mass loss rates, $\dot{M}_{\mathrm{rampressure}}$ and $\dot{M}_{\mathrm{tidal}}$, are computed using the selected methods (see §12.40 and §12.58 respectively). Finally, stellar luminosities evolve according to:

$$\dot{L}_{\mathrm{disk,stars}} = \Psi \mathcal{L}_\lambda(t_0 - t, Z_{\mathrm{gas}}) - \frac{L_{\mathrm{disk,stars}}}{\tau_{\mathrm{bar}}} - \frac{L_{\mathrm{disk,stars}}}{M_{\mathrm{disk,gas}} + M_{\mathrm{disk,stars}}} \dot{M}_{\mathrm{tidal}}, \tag{11.62}$$

where $\mathcal{L}_\lambda(t, Z)$ is the luminosity-to-mass ratio in the relevant band for a stellar population of age t and metallicity Z. The fraction of mass retained in the disk, f_{retained}, is given by:

$$\dot{f}_{\mathrm{retained}} = -\frac{f_{\mathrm{retained}}}{\tau_{\mathrm{bar}}}, \tag{11.63}$$

where the initial value of f_{retained} is arbitrary as we will be interested only in ratios of the quantity.

Integral Evolution

The standard disk component supports solving for the stellar luminosities as inactive variables, if the parameter `[diskLuminositiesStellarInactive]=true`. Across a timestep t_i to t_{i+1} the change in luminosity can be written as the following integral:

$$\Delta L_{\mathrm{disk,stars}} = \int_{t_i}^{t_{i+1}} \mathrm{d}t \Psi \mathcal{L}_\lambda(t_0 - t, Z_{\mathrm{gas}}) \frac{f_{\mathrm{retained}}(t_{i+1})}{f_{\mathrm{retained}}(t)} + L_{\mathrm{disk,stars}}(t_i) \frac{\dot{f}_{\mathrm{retained}}(t)}{f_{\mathrm{retained}}(t_i)}, \tag{11.64}$$

$$\Delta L_{\mathrm{spheroid,stars}} = \int_{t_i}^{t_{i+1}} \mathrm{d}t \Psi \mathcal{L}_\lambda(t_0 - t, Z_{\mathrm{gas}}) \left(1 - \frac{f_{\mathrm{retained}}(t_{i+1})}{f_{\mathrm{retained}}(t)} \right) - L_{\mathrm{disk,stars}}(t_i) \frac{\dot{f}_{\mathrm{retained}}(t)}{f_{\mathrm{retained}}(t_i)} \tag{11.65}$$

$$\tag{11.66}$$

The first term in the integral for disk stellar luminosity is just the usual production of starlight due to star formation, reduced by a factor $f_{\mathrm{retained}}(t_{i+1})/f_{\mathrm{retained}}(t)$ to account for that which will be retained in the disk at the end of the timestep. The second term accounts for transfer of starlight produced at $t < t_i$, since it integrates to $L_{\mathrm{disk,stars}}(t_i)(f_{\mathrm{retained}}(t_{i+1})/f_{\mathrm{retained}}(t_i) - 1)$ which is the change in the starlight of the disk over the timestep. The functions $f_{\mathrm{retained}}(t)$ and $M_{\mathrm{stars,disk,formed}}(t)$ have already been solved for during the differential evolution phase. Therefore, $f_{\mathrm{retained}}(t)$ and $\dot{f}_{\mathrm{retained}}(t)$ are available for this calculation. Furthermore, we can make the replacement $\Psi = \dot{M}_{\mathrm{stars,disk,formed}}(t)$ in the above, allowing us to avoid recomputing Ψ directly in these integral. The integral for the spheroid component simply reflects that starlight not retained in the disk is transferred to the spheroid.

Event Evolution

Node mergers: None

Satellite merging: Disks may be destroyed (or, potentially, created or otherwise modified) as the result of a satellite merging event, as dictated by the selected merger remnant mass movement method (see §12.18.1).

Node promotion: None

Additional Output

If the `[diskOutputStarFormationRate]` input parameter is set to true, then the instantaneous star formation rate in the disk (in units of $M_\odot$ Gyr^{-1}) will be included in the output, as `diskStarFormationRate`.

Structure

The radial size of the disk is found solving for equilibrium (i.e. the radius is such that the angular momentum of material at that radius is sufficient to provide rotational support) at the specified `[diskStructureSolverRadius]` which is given in units of the disk scale length. In converting from the mean specific angular momentum of the disk to the angular momentum at that radius, a flat rotation curve is assumed, i.e.:

$$j(r)/\langle j \rangle = rV \Bigg/ \frac{\int_0^\infty 2\pi r' \Sigma(r') r' V \mathrm{d}r'}{\int_0^\infty 2\pi r' \Sigma(r') \mathrm{d}r'} \,. \tag{11.67}$$

The option `[diskRadiusSolverCole2000Method]`, if set to `true`, alters this behavior to match that of the structure solver used by Cole et al. [2000], in which adiabatic contraction of the dark matter halo is solved for assuming that the disk has a spherical mass dsitribution. The specific angular momentum passed to the structure solver will be modified as follows in this case:

$$j(r) \rightarrow \left[j^2(r) - \left(V^2_\mathrm{disk}(r) r^2 - \mathrm{G} M_\mathrm{disk}(<r) r \right) \right]^{1/2} , \tag{11.68}$$

where V_disk is the rotation curve in the plane of the disk. This adjustment accounts for the difference between a thin disk and spherical mass distribution. Note that in this case (as in Cole et al. 2000) the resulting disk will not precisely satisfy $j(r) = rV_\mathrm{c}(r)$ where $V_\mathrm{c}(r)$ is the net rotation curve.

11.5. Galactic Spheroid

11.5.1. "Very Simple" Implementation

This implementation assumes a spheroid consisting of just gas and stellar masses, plus a half-mass radius (with no other assumption about structure being made).

Properties

The very simple galactic spheroid implementation defines the following properties:

`massGas` The mass of gas in the spheroid: $M_\mathrm{spheroid,gas}$ [`spheroidMassGas`];

`massStellar` The mass of stars in the spheroid: $M_\mathrm{spheroid,stars}$ [`spheroidMassStellar`].;

`radius` The half-mass radius of the sphreoid: $r_\mathrm{spheroid,1/2}$ [`spheroidRadius`].

Initialization

No initialization is performed—spheroids are created as needed.

Differential Evolution

Spheroid radii are computed using the selected galactic structure solver. In the very simple galactic spheroid implementation the gas mass evolves as:

$$\dot{M}_{\rm spheroid,gas} = \dot{M}_{\rm cooling} - \dot{M}_{\rm outflow,spheroid} - \dot{M}_{\rm stars,spheroid}, \tag{11.69}$$

where the rate of change of stellar mass is

$$\dot{M}_{\rm stars,spheroid} = \Psi - \dot{R}, \tag{11.70}$$

where $\dot{R}$ is the rate of mass recycling from stars, and

$$\Psi = \frac{M_{\rm spheroid,gas}}{\tau_{\rm spheroid,star\ formation}} \tag{11.71}$$

with $\tau_{\rm spheroid,star\ formation}$ being the greater of the star formation timescale and $\Gamma_{\rm spheroid,starformation,minimum}\tau_{\rm dyn}$, where $\tau_{\rm dyn}$ is the dynamical time of the halo, and $\Gamma_{\rm spheroid,starformation,minimum}$ =`[spheroidStarFormationTimescaleMinimum]`. The outflow rate, $\dot{M}_{\rm outflow,spheroid}$, is computed for the current star formation rate and gas properties by the prescriptions for non-expulsive supernova feedback (see §12.52), but is limited to a maximum of $M_{\rm spheroid,gas}/\Gamma_{\rm spheroid,outflow,minimum}\tau_{\rm dyn}$, where $\Gamma_{\rm spheroid,outflow,minimum}$ =`[spheroidOutflowTimescaleMinimum]`. This outflow is piped to the hot halo component.

Event Evolution

Node mergers: None

Satellite merging: Spheroids may be destroyed (or, potentially, created or otherwise modified) as the result of a satellite merging event, as dictated by the selected merger remnant mass movement method (see §12.18.1).

Node promotion: None

11.5.2. "Standard" Implementation

The standard spheroid implementation assumes a spheroid density profile described by a single length scale in which stars trace gas. Currently, two options for the density profile are allowed[5]:

`hernquist` Assumes a Hernquist profile [Hernquist, 1990] for the spheroidal component of a galaxy.

`sersic` Assumes a Sérsic profile (Sérsic 1963; see also Mazure and Capelato 2002) for the spheroidal component of a galaxy in which stars trace gas. The projected density profile of the spheroid is given by:

$$\Sigma(R) \propto \exp\left(-b_{\rm n} R^{1/n}\right), \tag{11.72}$$

where the Sérsic index, n =`[spheroidSersicIndex]` and the coefficient $b_{\rm n} = 2.303(0.8689n - 0.1447)$ Wadadekar et al. [1999]. The 3D density distribution for a given n is inferred by solving the relevant inverse Abel integral.

[5] The spheroid density distribution is handled internally using a `massDistribution` object (see §16.4.1). As such, any mass distribution implemented as an extension of the `massDistribution` class (and which is described by a single length scale) could be trivially addded to the standard spheroid component.

Properties

The standard galactic spheroid implementation defines the following properties:

`masGass` The mass of gas in the spheroid: $M_{\rm spheroid,gas}$ [`spheroidMassGas`].

`abundancesGas` The mass of elements in the gaseous spheroid: $M_{Z,\rm spheroid,gas}$ [`spheroidAbundancesGas{abundanceName}`].

`massStellar` The mass of stars in the spheroid: $M_{\rm spheroid,stars}$ [`spheroidMassStellar`].

`abundancesStellar` The mass of elements in the stellar spheroid: $M_{Z,\rm spheroid,stars}$ [`spheroidAbundancesStellar{abundancel`

`luminositiesStellar` The luminosities (in multiple bands) of the stellar spheroid: $L_{\rm spheroid,stars}$ [`spheroidLuminositiesSte`

`angularMomentum` The pseudo-angular momentum[6] of the spheroid, $J_{\rm spheroid}$ [`spheroidAngularMomentum`]. The parameter [`spheroidAngularMomentumAtScaleRadius`] controls the ratio of the specific pseudo-angular momentum at the scale radius of the standard spheroid to the mean specific pseudo-angular momentum. By default, this parameter is set to [`spheroidAngularMomentumAtScaleRadius`] $= I_2/I_3$, where

$$I_n = \int_0^\infty \rho(r) r^n {\rm d}r, \tag{11.73}$$

and $\rho(r)$ is the spheroid density profile, which is appropriate for a flat rotation curve. In some cases (e.g. the Hernquist profile) one or both of I_2 and I_3 can be infinite. In such cases [`spheroidAngularMomentumAtScaleRa`] $= 0.5$ is assumed by default. If a finite truncation radius is assumed, or a different rotation curve is assumed, this ratio may be finite. The [`spheroidAngularMomentumAtScaleRadius`] parameter allows control over these assumptions.

`radius` The radial scale length of the spheroid, $r_{\rm spheroid}$ [`spheroidRadius`].

`velocity` The circular velocity of the spheroid at $r_{\rm spheroid}$, $V_{\rm spheroid}$ [`spheroidVelocity`].

`massStellarFormed` The total mass of stars ever formed in the spheroid: $M_{\rm spheroid,stars,formed}$.

and the following pipes:

`energyInput` Energy sent through this pipe is added to the gas of the spheroid and will result in an outflow (see below). Input energy should be in units of $M_\odot$ km^2 s^{-2} Gyr^{-1} and must be positive (energy cannot be removed from the gas via this pipe).

`massGasSink` Removes gas (and proportionate amounts of angular momentum and elements) from the spheroid gas. Removed mass should be in units of $M_\odot$ and must be positive (a negative mass sink would add mass to the spheroid which is not allowed via this pipe).

Initialization

No initialization is performed—spheroids are created as needed.

[6] Effectively the angular momentum that the spheroid would have, were it rotationally supported rather than pressure supported.

Differential Evolution

In the standard galactic spheroid implementation the gas mass evolves as[7]:

$$\dot{M}_{\rm spheroid,gas} = -\dot{M}_{\rm outflow,spheroid} - \dot{M}_{\rm stars,spheroid} - \dot{M}_{\rm rampressure} - \frac{M_{\rm spheroid,gas}}{M_{\rm spheroid,gas}+M_{\rm spheroid,stars}}\dot{M}_{\rm tidal}, \tag{11.74}$$

where the rate of change of stellar mass is

$$\dot{M}_{\rm stars,spheroid} = \Psi - \dot{R} - \frac{M_{\rm spheroid,stars}}{M_{\rm spheroid,gas}+M_{\rm spheroid,stars}}\dot{M}_{\rm tidal}, \tag{11.75}$$

$$\dot{M}_{\rm stars,disk,formed} = \Psi, \tag{11.76}$$

with

$$\Psi = \frac{M_{\rm spheroid,gas}}{\tau_{\rm spheroid,star\ formation}} \tag{11.77}$$

with $\tau_{\rm spheroid,star\ formation}$ being the star formation timescale and $\dot{R}$ is the rate of mass recycling from stars. If `[spheroidStarFormationInSatellites]=true` then the star formation rate is forced to zero in satellite galaxies. Element abundances (including total metals) evolve according to:

$$\dot{M}_{Z,\rm spheroid,gas} = -\dot{M}_{Z,\rm outflow,spheroid} - \dot{M}_{Z,\rm stars,spheroid} + \dot{y} - \frac{M_{Z,\rm spheroid,gas}}{M_{\rm spheroid,gas}}\dot{M}_{\rm rampressure} - \frac{M_{Z,\rm spheroid,gas}}{M_{\rm spheroid,gas}+M_{\rm spheroid,stars}}\dot{M}_{\rm tidal}, \tag{11.78}$$

and

$$\dot{M}_{Z,\rm stars,spheroid} = \Psi\frac{M_{Z,\rm spheroid,gas}}{M_{\rm spheroid,gas}} - \dot{R}_Z - \frac{M_{Z,\rm spheroid,stars}}{M_{\rm spheroid,gas}+M_{\rm spheroid,stars}}\dot{M}_{\rm tidal} \tag{11.79}$$

where $\dot{y}$ is the rate of element yield from stars and $\dot{R}_Z$ is the rate of element recycling. The angular momentum evolves as:

$$\dot{J}_{\rm spheroid} = -(\dot{M}_{\rm outflow,spheroid}+\dot{M}_{\rm rampressure}+\dot{M}_{\rm tidal})\frac{J_{\rm spheroid}}{M_{\rm spheroid,gas}+M_{\rm spheroid,stars}} + |\mathcal{T}|(M_{\rm spheroid,gas}+M_{\rm spheroid,stars})R^2_{\rm spheroid}. \tag{11.80}$$

The outflow rate, $\dot{M}_{\rm outflow,spheroid}$, is computed for the current star formation rate and gas properties by the stellar properties subsystem (see §12.47) and prescriptions for expulsive and non-expulsive supernova feedback (see §12.53 and §12.52 respectively), with an additional contribution given by

$$\dot{M}_{\rm outflow,spheroid} = \beta_{\rm spheroid,energy}\frac{\dot{E}_{\rm gas,spheroid}}{V^2_{\rm spheroid}} \tag{11.81}$$

where $\beta_{\rm spheroid,energy}$ =`[spheroidEnergeticOutflowMassRate]` is an input parameter, and $\dot{E}_{\rm gas,spheroid}$ is any input energy sent through the `Tree_Node_Spheroid_Gas_Energy_Input` pipe, but is not allowed to exceed $M_{\rm gas,spheroid}/\alpha_{\rm outflowminimum,spheroid}\tau_{\rm spheroid,dynamical}$, where $\tau_{\rm spheroid,dynamical} = R_{\rm spheroid}/V_{\rm spheroid}$ is the dynamical time of the spheroid and $\alpha_{\rm outflowminimum,spheroid}$ =`[spheroidOutflowTimescaleMinimum]` is the shortest timescale (in units of the dynamical timescale) on which gas can be removed from the

[7]There may be an additional contribution to the mass and angular momentum rates of change in the spheroid due to material transferred from the disk component via the bar instability mechanism (see §11.4.3). This is not included here as it is not intrinsic to this specific spheroid implementation—it is handled explicitly by the disk component and so applies equally to any spheroid component implementation.

spheroid. This limit prevents the spheroid being depleted on arbitrarily short timescales. The non-expulsive component of the outflow is piped to the hot halo component. The ram pressure and tidal mass loss rates, $\dot{M}_{\rm rampressure}$ and $\dot{M}_{\rm tidal}$, are computed using the selected methods (see §12.40 and §12.58 respectively). The final term in 11.80 accounts for tidal heating of the spheroid due to the tidal field, $\mathcal{T}$.

Finally, stellar luminosities evolve according to:

$$\dot{L}_{\rm spheroid,stars} = \Psi \mathcal{L}_\lambda(t_0 - t, Z_{\rm gas}) - \frac{L_{\rm spheroid,stars}}{M_{\rm spheroid,gas} + M_{\rm spheroid,stars}} \dot{M}_{\rm tidal}, \tag{11.82}$$

where $\mathcal{L}_\lambda(t, Z)$ is the luminosity-to-mass ratio in the relevant band for a stellar population of age t and metallicity Z.

Integral Evolution

The standard spheroid component supports solving for the stellar luminosities as inactive variables, if the parameter `[spheroidLuminositiesStellarInactive]=true`. Across a timestep t_i to t_{i+1} the change in luminosity can be written as the following integral:

$$\Delta L_{\rm spheroid,stars} = \int_{t_i}^{t_{i+1}} {\rm d}t \Psi \mathcal{L}_\lambda(t_0 - t, Z_{\rm gas}), \tag{11.83}$$

which is just the usual production of starlight due to star formation. As the function $M_{\rm stars,spheroid,formed}(t)$ has already been solved for during the differential evolution phase we can make the replacement $\Psi = \dot{M}_{\rm stars,disk,formed}(t)$ in the above, allowing us to avoid recomputing Ψ directly in these integral.

Event Evolution

Node mergers: None

Satellite merging: Spheroids may be created as the result of a satellite merging event, as dictated by the selected merger remnant mass movement method (see §16.4.1).

Node promotion: None.

Additional Output

If the `[spheroidOutputStarFormationRate]` input parameter is set to true, then the instantaneous star formation rate in the spheroid (in units of $M_\odot$ Gyr^{-1}) will be included in the output, as `spheroidStarFormationRate`.

11.6. Host History

This component class tracks various properties of the host halo of each node over time.

11.6.1. "Standard" Implemenation

Properties

The standard host history implementation defines the following properties:

`hostMassMaximum` The maximum mass of the host halo in which this node has ever been hosted: $M_{\rm host,maximum}$ [`hostHistoryHostMassMaximum`].

Initialization

The maximum host mass is initialized to the initial host mass, or to -1 for initially isolated nodes.

Differential Evolution

N/A.

Event Evolution

Post evolution: Immediately after evolution of the node, the maximum host mass is set to the maximum of the current host mass and the previous maximum host mass.

Node mergers: As for "post evolution".

Satellite merging: None.

Node promotion: As for "post evolution".

11.7. Mass Flow Statistics

This component class tracks the flow of mass between different components of a node.

11.7.1. "Standard" Implemenation

Properties

The standard mass flow statistics implementation defines the following properties:

`cooledMass` The cumulative mass of has which has cooled directly onto this galaxy: $M_{\rm cooled}$ [`massFlowStatisticsCooledMass`

Initialization

The cooled mass is initialized to zero.

Differential Evolution

The cooled mass increases at a rate equal to the cooling rate onto the galaxy:

$$\dot{M}_{\rm cooled} = \dot{M}_{\rm cool}. \tag{11.84}$$

Event Evolution

Post evolution: None.

Node mergers: None.

Satellite merging: None.

Node promotion: None.

11.8. Basic Properties

Basic properties are the total mass of a node and the cosmic time at which it currently exists.

11.8.1. "Non-evolving" Implemenation

Properties

The non-evolving basic properties implementation defines the following properties:

`mass` The total mass of the node: $M_{\rm node}$ [`basicMass`].

`time` The time at which the node is defined: $t_{\rm node}$.

`timeLastIsolated` The time at which the node was last an isolated halo (i.e. not a subhalo): [`basicTimeLastIsolated`].

Initialization

All basic properties are required to be initialized by the merger tree construction routine.

Differential Evolution

Properties are evolved according to:

$$\dot{M}_{\rm node} = 0 \tag{11.85}$$
$$\dot{t}_{\rm node} = 1. \tag{11.86}$$

Event Evolution

Node mergers: None.

Satellite merging: None.

Node promotion: $M_{\rm node}$ is updated to the node mass of the parent prior to promotion.

11.8.2. "Standard Implemenation

Properties

The standard basic properties implementation defines the following properties:

`mass` The total mass of the node: $M_{\rm node}$ [`basicMass`].

`time` The time at which the node is defined: $t_{\rm node}$.

`timeLastIsolated` The time at which the node was last an isolated halo (i.e. not a subhalo): [`basicTimeLastIsolated`].

Initialization

All basic properties are required to be initialized by the merger tree construction routine.

Differential Evolution

Properties are evolved according to:

$$\dot{M}_{\rm node} = \begin{cases} \frac{M_{\rm node,parent}-M_{\rm node}}{t_{\rm node,parent}-t_{\rm node}} & \text{if primary progenitor} \\ 0 & \text{otherwise,} \end{cases} \tag{11.87}$$

$$\dot{t}_{\rm node} = 1, \tag{11.88}$$

where the "parent" subscript indicates a property of the parent node in the merger tree.

Event Evolution

Node mergers: None.

Satellite merging: None.

Node promotion: $M_{\rm node}$ is updated to the node mass of the parent prior to promotion.

11.8.3. "Standard-Tracking Implemenation

Properties

The standard-tracking basic properties implementation extends the standard implementation and defines the following additional properties:

`massMaximum` The maximum total mass of the node achieved prior to the current time along this node's branch: $M_{\rm node,maximum}$ [`basicMassMaximum`].

Initialization

The maximum mass along each branch is computed from the merger tree structure at initialization time.

Differential Evolution

None.

Event Evolution

Node mergers: None.

Satellite merging: None.

Node promotion: $M_{\rm node,maximum}$ is updated to the node maximum mass of the parent prior to promotion.

11.8.4. "Standard-Extended Implemenation

Properties

The standard-extended basic properties implementation extends the standard implementation and defines the following additional properties:

`massBertschinger` The "Bertschinger" mass of the node, defined as the mass enclosing a density contrast equal to that for the spherical collapse model: $M_{\rm node,Bertschinger}$ [`basicMassBertschinger`].

`accretionRateBertschinger` The accretion rate of "Bertschinger" mass onto the node: $\dot{M}_{\rm node,Bertschinger}$ [`basicAccretionRateBertschinger`].

`radiusTurnaround` The turnaround radius corresponding to the current "Bertshcinger"mass: $R_{\rm ta}$ [`basicRadiusTurnaround`]

Initialization

The Bertschinger mass and accretion rate for each node are computed from $M_{\rm basic}$ and the dark matter density profile. The turnaround radius is computed from the virial radius (under the appropriate definition for the Bertschinger mass) and the ratio of turnaround to virial radius from spherical collapse models. The specific spherical collapse model to use is determined by the `[nodeComponentBasicExtendedSphericalCollapseType]` parameter—a value of "`matterLambda`" causes the solution for matter+cosmological constant universes to be used, while a value of "`matterDarkEnergy`" causes the solution for matter+dark energy universes to be used.

Differential Evolution

None.

Event Evolution

Node mergers: The accretion rate of Bertschinger mass is set to zero.

Satellite merging: None.

Node promotion: $\dot{M}_{\rm node,Bertschinger}$ is updated to the node Bertschinger accretion rate of the parent prior to promotion.

11.9. Position

The position component implements the position and velocity of each galaxy. See §10.3 for important notes on velocity definitions in Galacticus.

11.9.1. "Preset" Implemenation

Properties

The preset position implementation defines the following properties:

`position` The 3-D position of the node: $\mathbf{x}$ [`positionPosition[X|Y|Z]`].

`velocity` The 3-D velocity of the node: $\mathbf{v}$ [`positionVelocity[X|Y|Z]`].

`positionHistory` The history of the node's position in 6-D phase space, usually used for satellite nodes.

Initialization

None—all properties are assumed to have been preset, usually by the merger tree construction routine.

Differential Evolution

None. Positions and velocities do not evolve for a given node. When output, if a 6-D position history is available than the position and velocity from the history entry closest to the output time will be used[8].

Event Evolution

Node mergers: If `positionsPresetSatelliteToHost=true` then the position and velocity of the satellite node is set equal to that of the host node (this is useful if position data is not available for orphaned halos for example, which would otherwise remain fixed in physical coordinates at their last known position—the position/velocity will also be updated to that of the new host each time a satellite's host changes), otherwise, none.

Satellite merging: None.

Node promotion: The position and velocity are updated to those of the parent node.

11.10. Satellite Orbit

This component tracks the orbital properties of subhalos.

11.10.1. "Preset" Implementation

Properties

The preset satellite orbit implementation defines the following properties:

`mergTime` The time until the satellite will merge with its host: $t_{\rm satellite,merge}$ [`satelliteMergeTime`].

`timeOfMerging` The cosmological time at which the satellite will merge with its host: $T_{\rm satellite,merge}$.

`boundMass` The remaining, total bound mass of the satellite (this property is read only—it is determined from the `boundMassHistory` property).

`boundMassHistory` A history time-series of the total bound mass of the satellite.

`virialOrbit` The orbit (a `keplerOrbit` object; see §16.5.1) of the satellite at virial orbit crossing.

Note that the `mergeTime` and `timeOfMerging` effectively provide the same information. For that reason, setting one of them will automatically set the other accordingly.

Initialization

None. This method assumes that merging times and bound mass histories will be set externally (usually when the merger tree is constructed).

Differential Evolution

None.

[8] While interpolation could be used this is usually a bad idea. For nodes that are satellites in a halo for example, no simple interpolation algorithm can correctly account for the complex orbital dynamics by which the position and velocity is actually evolving.

Event Evolution

Node mergers: None.

Satellite merging: None.

Node promotion: None.

11.10.2. "Very Simple" Implementation

Properties

The simple satellite orbit implementation defines the following properties:

`mergeTime` The time until the satellite will merge with its host: $t_{\text{satellite,merge}}$ [`satelliteMergeTime`].

Initialization

None.

Differential Evolution

Properties are evolved according to:

$$\dot{t}_{\text{satellite,merge}} = -1. \tag{11.89}$$

Event Evolution

Node mergers: The component is created and the time to merging is assigned a value.

Satellite merging: None.

Node promotion: Not applicable (component only exists for satellite nodes).

11.10.3. "Simple" Implementation

Properties

The simple satellite orbit implementation defines the following properties:

`mergeTime` The time until the satellite will merge with its host: $t_{\text{satellite,merge}}$ [`satelliteMergeTime`].

`boundMass` The remaining, total bound mass of the satellite: $M_{\text{node,bound}}$ [`satelliteBoundMass`].

`virialOrbit` The orbit (returned as a `keplerOrbit` object; see §16.5.1) of the satellite at the point of virial radius crossing.

Initialization

None.

Differential Evolution

Properties are evolved according to:

$$\dot{t}_{\rm satellite,merge} = -1, \tag{11.90}$$

with $\dot{M}_{\rm node,bound}$ set to the rate given by the `darkMatterHaloMassLossRateMethod` method (see §16.4.1). The virial orbit is a fixed quantity and does not evolve.

Event Evolution

Node mergers: The component is created and the time to merging is assigned a value. The bound mass is set to the current total mass of the node. If `satelliteOrbitStoreOrbitalParameters=true` then a virial orbit is selected (unless one has already been set for the node) using the `virialOrbitsMethod` (see §12.51.2) and stored (otherwise, a new virial orbit will be computed—possibly at random—each time the virial orbit is requested). If `[satelliteOrbitResetOnHaloFormation]=true` then satellite orbits will be reset on halo formation events (see §11.16).

Satellite merging: None.

Node promotion: Not applicable (component only exists for satellite nodes).

11.10.4. "Orbiting" Implementation

Properties

The orbiting satellite orbit implementation defines the following properties:

`position` The 3-dimensional position of the satellite relative to its host: $\mathbf{r} = (x, y, z)$.

`velocity` The 3-dimensional velocity of the satellite relative to its host: $\mathbf{v} = (v_x, v_y, v_z)$.

`mergeTime` The time until the satellite will merge with its host: $t_{\rm satellite,merge}$ [`satelliteMergeTime`].

`boundMass` The remaining, total bound mass of the satellite: $M_{\rm node,bound}$ [`satelliteBoundMass`].

`virialOrbit` The orbit (returned as a `keplerOrbit` object; see §16.5.1) of the satellite at the point of virial radius crossing.

`tidalTensorPathIntegrated` The time integral of the tidal tensor along the orbit of the satellite from initialization: G_{ij}.

`tidalHeatingNormalized` The tidal heating energy per radius squared desposited into the satellite: $Q_{\rm tidal}$.

Initialization

The satellite position, if not already assigned, is selected such that its magnitude is set according to its virial orbit parameters and its direction is chosen at random assuming an isotropic distribution. The satellite velocity is similarly selected. The integrated tidal tensor, G_{ij}, is initialized to the null tensor, while $Q_{\rm tidal}$ is initialized to zero.

Differential Evolution

Properties are evolved according to:

$$\dot{\mathbf{r}} = \mathbf{v} \tag{11.91}$$

$$\dot{\mathbf{v}} = -\frac{GM_{\rm host}(<r)\mathbf{r}}{r^3}\left(1+\frac{M_{\rm node,bound}}{M_{\rm host}(<r)}\right)+\mathbf{a}_{\rm DF} \tag{11.92}$$

$$\dot{G}_{ij} = g_{ij} - G_{ij}/T_{\rm orb}, \tag{11.93}$$

with $r = |\mathbf{r}|$, $\mathbf{a}_{DF}$ set to the rate given by the `satelliteDynamicalFrictionMethod` (see §12.41), g_{ij} being the tidal tensor, $T_{\rm orb}$ being the satellite's orbital period, $\dot{M}_{\rm node,bound}$ set to the rate given by the `satelliteTidalStrippingMethod` (see §12.43), and $\dot{Q}_{\rm tidal}$ set to the rate given by the `satelliteTidalHeatingMethod` (see §12.44). Note that in the evolution of G_{ij} a decay term is artificially introduced such that the integration of g_{ij} is effectively over just the previous orbit (i.e. over the previous tidal shock).

Event Evolution

Node mergers: The component is created and the time to merging is given a value of -1 to indicate that the satellite is not about to merge as unmerged. Once the satellite satisfies one of the two merging conditions:

$$r < R_{\rm host} + R_{\rm satellite} \tag{11.94}$$

$$M_{\rm node,bound} < f_{\rm d} M_{\rm node,basic}, \tag{11.95}$$

with R being the node's half-mass radius, $M_{\rm node,basic}$ being the node's initial mass, and $f_{\rm d}$ =`[satelliteOrbitingDestruction` the node is considered merged and the time to merging is set to zero. The bound mass is set to the current total mass of the node. A virial orbit is selected using the `virialOrbitsMethod` (see §12.51.2).

Satellite merging: None.

Node promotion: Not applicable (component only exists for satellite nodes).

11.11. N-body Halo Properties

This component is intended for tracking of miscellaneous properties derived from N-body simulation merger trees.

11.11.1. "Standard" Implementation

Properties

The standard N-body component implementation defines the following properties:

`particleCount` The number of particles in the N-body halo [`nbodyParticleCount`].

`velocityMaximum` The maximum of the N-body halo's rotation curve (in units of km/s) [`nbodyVelocityMaximum`].

`velocityDispersion` The velocity dispersion of the N-body halo (in units of km/s) [`nbodyVelocityDispersion`].

Initialization

The N-body properties of each node are assumed to have been preset prior to merger tree initialization. They are assumed to remain constant along each branch of the tree.

Differential Evolution

None.

Event Evolution

Node mergers: None.

Satellite merging: None.

Node promotion: The properties are updated to equal those of the parent node.

11.12. Dark Matter Halo Spin

11.12.1. "Random" Implementation

Properties

The random dark matter halo spin implementation defines the following properties:

`spin` The spin parameter of the halo: λ [`spinSpin`].

Initialization

The spin parameter of each node, if not already assigned, is selected at random from a distribution of spin parameters. This value is assigned to the earliest progenitor of the halo traced along its primary branch. The value is then propagated forward along the primary branch until the node mass exceeds that of the node for which the spin was selected by a factor of `[randomSpinResetMassFactor]`, at which point a new spin is selected at random, and the process repeated until the end of the branch is reached.

Differential Evolution

The spin parameter does not evolve.

Event Evolution

Node mergers: None.

Satellite merging: None.

Node promotion: The spin is updated to equal that of the parent node. (The two will differ only if this is a case where the new halo node was sufficiently more massive than the node for which a spin was last selected that a new spin value was chosen.)

11.12.2. "Preset" Implementation

Properties

The preset dark matter halo spin implementation defines the following properties:

`spin` The spin parameter of the halo: λ [`spinSpin`].

`spinGrowthRate` The growth rate spin parameter of the halo (in units of Gyr^{-1}).

Initialization

The spin parameter of each node is assumed to have been preset prior to merger tree initialization. The growth rate is computed assuming linear growth with time along each branch.

Differential Evolution

The spin parameter evolves linearly with time between node and parent node.

Event Evolution

Node mergers: None.

Satellite merging: None.

Node promotion: The spin and growth rate are updated to equal those of the parent node.

11.12.3. "Preset3D" Implementation

The type extends the `preset` spin component implementation, and so provides all properties of that implementation plus those described below.

Properties

The preset dark matter halo spin implementation defines the following properties:

`spinVector` The spin vector of the halo: λ [`spinSpinVector`].

`spinVectorGrowthRate` The growth rate of the spin vector of the halo (in units of Gyr^{-1}).

Initialization

The spin vector of each node is assumed to have been preset prior to merger tree initialization. The growth rate is comptued assuming linear growth with time along each branch.

Differential Evolution

The spin parameter evolves linearly with time between node and parent node.

Event Evolution

Node mergers: None.

Satellite merging: None.

Node promotion: The spin vector and growth rate are updated to equal those of the parent node.

11.12.4. "Vitvitska Implementation

Properties

The Vitvitska dark matter halo spin implementation follows the Vitvitska et al. [2002] model for halo spins, and defines the following properties:

`spin` The spin parameter of the halo: λ [`spinSpin`].

Initialization

The spin parameters of nodes having no children are drawn at random from the selected spin distribution function. For other nodes, their angular momentum is set equal to the sum of the spin angular momentum of their primary progenitor and the orbital angular momenta of all non-primary progenitors (which are determined from the orbital parameters) modulated by a factor dependent on the mass ratio of the merging halos:

$$\mathbf{J}^{(\mathrm{spin})}_{\mathrm{parent}} = \mathbf{J}^{(\mathrm{spin})}_{1} + \sum_{i=2}^{N} \left(1 + \frac{m_i}{m_1}\right)^{-\alpha} \mathbf{J}^{(\mathrm{orbit})}_{i}. \tag{11.96}$$

where $\alpha =$ [`spinVitvitskaMergerRatioExponent`].

Differential Evolution

The spin parameter does not evolve.

Event Evolution

Node mergers: None.

Satellite merging: None.

Node promotion: None.

11.13. Dark Matter Profile

This component stores dynamic properties associated with dark matter halo density profiles.

11.13.1. "Scale" Implementation

Properties

The scale dark matter profile implementation defines the following properties:

`scale` The scale length of the density profile [`darkMatterProfileScale`];

`scaleGrowthRate` The growth rate of the scale length of the density profile.

Initialization

The scale length of each node, if not already assigned, is assigned using the concentration parameter function (see §12.11.3), but is not allowed to drop below `[darkMatterProfileMinimumConcentration]`, such that the scale length is equal to the virial radius divided by that concentration.

Differential Evolution

The scale radius grows linearly with time to interpolate between the scale radii of child and parent halos.

Event Evolution

Node mergers: None.

Satellite merging: None.

Node promotion: None.

11.13.2. "Scale Preset" Implementation

Properties

The "scale preset" dark matter profile implementation defines the following properties:

`scale` The scale length of the density profile [`darkMatterProfileScale`];

`scaleGrowthRate` The growth rate of the scale length of the density profile.

Initialization

The scale length of each node is assumed to be assigned udirng tree construction.

Differential Evolution

The scale radius grows linearly with time to interpolate between the scale radii of child and parent halos.

Event Evolution

Node mergers: None.

Satellite merging: None.

Node promotion: None.

11.13.3. "Scale+Shape" Implementation

Properties

The scale+shape dark matter profile implementation defines the following properties:

`scale` The scale length of the density profile [`darkMatterProfileScale`];

`scaleGrowthRate` The growth rate of the scale length of the density profile.

`shape` A shape parameter describing the density profile [`darkMatterProfileShape`];

`shapeGrowthRate` The growth rate of the shape parameter of the density profile.

Initialization

The scale length of each node, if not already assigned, is assigned using the concentration parameter function (see §12.11.3), but is not allowed to drop below `[darkMatterProfileMinimumConcentration]`, such that the scale length is equal to the virial radius divided by that concentration. The shape parameter of each node is assigned using the dark matter profile shape function (see §16.4.1).

Differential Evolution

The scale radius and shape parameter of each node grow linearly with time to interpolate between the scale radii and shape parameters of child and parent halos.

Event Evolution

Node mergers: None.

Satellite merging: None.

Node promotion: None.

11.14. Merging Statistics

This component records statistics associated with galaxy merging.

11.14.1. "Standard" Implementation

Properties

The standard merging statistics implementation defines the following properties:

`galaxyMajorMergerTime` The time of the last major merger associated with this galaxy, output as the time *since* the last major merger [`mergingStatisticsGalaxyMajorMergerTime`];

`nodeMajorMergerTime` The time of the last major merger (as defined by the `[nodeMajorMergerFraction]` parameter) between this node and another, output as the time *since* the last major merger [`mergingStatisticsNodeMajo`

`nodeFormationTime` The time at which the node is judged to have "formed", defined as the time at which its main branch progenitor had a mass equal to a fraction `[nodeFormationMassFraction]` of the node's initial mass in the merger tree [`mergingStatisticsNodeFormationTime`];

`nodeHierarchyLevel` The depth of this node in the initial merger tree hierarchy. For example, a node on the main branch has level 0. A node which merges directly onto the main branch has level 1. A node that merges onto a node that merges directly onto the main branch has level 2, etc. [`mergingStatisticsNodeHierarchyLevel`];

Initialization

The times of the last mergers are stored each time a major merger occurrs and this component is created (if necessary) the first time such a merger occurs. Formation times and hierarchy levels are computed during merger tree initialization.

Differential Evolution

The times of the last mergers do not evolve.

Event Evolution

Node mergers: The time of the last node merger in the parent node is reset to the current time if the merger is major.

Satellite merging: The time of the last merger is reset to the current time if the merger is major.

Node promotion: The time of the last major node merger is updated to that of the parent if that time is more recent.

11.14.2. "Recent" Implementation

Properties

The recent merging statistics implementation defines the following properties:

`recentMajorMergerTime` The number of node major mergers (defined as a merger with mass ratio in excess of [`nodeMajorMergerFraction`]) occuring within a recent interval, Δt where $\Delta t =$ [`nodeRecentMajorMergerInterv` if [`nodeRecentMajorMergerIntervalType`]=`absolute`, or $\Delta t =$ [`nodeRecentMajorMergerInterval`] $\tau_{\rm dyn}$ (where $\tau_{\rm dyn}$ is the halo dynamical time) if [`nodeRecentMajorMergerIntervalType`]=`dynamical`. [`mergingStatisticsRecentMajorMergerTime`];

Initialization

The number of recent mergers for each output time are initialized to zero.

Differential Evolution

The number of recent mergers do not evolve.

Event Evolution

Node mergers: If the merger is major and occurs within Δt of an output time, the number of recent major mergers is increased by one.

Satellite merging: N/A.

Node promotion: N/A.

11.14.3. "Major" Implementation

Properties

The major merging statistics implementation defines the following properties:

`majorMergerTime` The time(s) of galaxy major mergers experience by this node. If non-empty, the merger times are output to a dataset `majorMergers/Output<O>/mergerTree<T>/node<N>` (where `<O>` is the output number, `<T>` is the tree index, and `<N>` is the node index) in the GALACTICUS output file.

Initialization

The times are initialized to an empty list.

Differential Evolution

N/A.

Event Evolution

Node mergers: N/A.

Satellite merging: If the merger is major, the time of the merger is appended to the list of merging times for the host node.

Node promotion: Times recorded from the parent node are appended to those of the child node prior to promotion.

11.15. Age Statistics

This component records statistics associated with galaxy or halo age.

11.15.1. "Standard" Implementation

Properties

The standard age statistics implementation defines the following properties:

`(disk|spheroid)TimeWeightedIntegratedSFR` A time-weighted integral over the star formation rate of the specified component [`ageStatistics(Disk|Spheroid)TimeWeightedIntegratedSFR`];

`(disk|spheroid)IntegratedSFR` A unweighted integral over the star formation rate of the specified component [`ageStatistics(Disk|Spheroid)IntegratedSFR`];

Initialization

The integrals are all initialized to zero.

Differential Evolution

The time-weighted integrals evolve as:

$$\frac{\mathrm{d}(M\tau)_\star}{\mathrm{d}t} = t\phi(t), \tag{11.97}$$

where t is cosmic time and $\phi(t)$ is the star formation rate in the component. The unweighted integrals evolve as:

$$\frac{\mathrm{d}M_\star}{\mathrm{d}t} = \phi(t). \tag{11.98}$$

The mean cosmic time of formation for the stars in either component can be found by taking the ratio of time-weighted to unweighted integrals. The mean age can then be found by subtracting the mean cosmic time of formation from the present time.

Event Evolution

Node mergers: N/A.

Satellite merging: The values of the integrals are removed from the merging galaxy and transferred to the appropriate component of the target galaxy. Additionally, in a major merger, the values of the integral are transferred between components in the target galaxy as dictated by the merging rules.

Node promotion: N/A.

11.16. Formation Times

This component implements "formation times" of dark matter halos.

11.16.1. "Cole2000" Implementation

Properties

The "Cole2000' formation times implementation defines the following properties:

`formationTime` The time at which the halo last "formed". Formation is defined as an increase in the mass of the halo by a factor `[haloReformationMassFactor]`.

Initialization

The formation time is set to the current time and this component is created the first time such a merger occurs.

Differential Evolution

The formation time does not evolve. When the node mass exceeds the mass at the formation time by a factor `[haloReformationMassFactor]` evolution is interrupted and the formation time reset to the current time.

Event Evolution

Node mergers: None.

Satellite merging: None.

Node promotion: None.

11.17. Indices

This component tracks various indices associated with nodes.

11.17.1. "Standard" Implementation

Properties

The "standard' indices implementation defines the following properties:

`branchTip` The index of the node at the tip of the branch to which the node belongs.

Initialization

The branch tip index is set to equal the index of each branch tip node in the tree.

Differential Evolution

The indices do not evolve.

Event Evolution

Node mergers: None.

Satellite merging: None.

Node promotion: None.

11.18. Inter-Output Quantities

This component tracks various quantities averaged between successive outputs.

11.18.1. "Standard" Implementation

Properties

The "standard' inter-output implementation defines the following properties:

`diskStarFormationRate` The mean star formation rate (in units of $M_\odot$ Gyr^{-1}) in the disk between the previous and current output times.

`spheroidStarFormationRate` The mean star formation rate (in units of $M_\odot$ Gyr^{-1}) in the spheroid between the previous and current output times.

Initialization

No initialization is carried out—component is created as needed.

Differential Evolution

The disk and spheroid star formation rates, $\langle \dot{M}_\star \rangle$, evolve according to:

$$\frac{\mathrm{d}}{\mathrm{d}t}\langle \dot{M}_\star \rangle = \frac{\dot{M}_\star}{\Delta t}, \tag{11.99}$$

where $\dot{M}_\star$ is the instantaneous star formation rate in the corresponding component and Δt is the time between the previous and next output times.

Event Evolution

Node mergers: None.

Satellite merging: The mean star formation rates from the satellite are added to those of the parent.

Node promotion: None.

Output: All quantities are zeroed after each output.

12. Physical Implementations

12.1. Analytic Solvers

Galacticus typically solves the system of ODEs which describe the evolution of a galaxy using a numerical solver. However, in some situations analytic solutions may be available. Galacticus allows for the use of analytic solvers in such cases. Currently available analytic solvers are described below.

12.1.1. Simple Disk Satellites

This solver, which can be activated by setting `[diskVerySimpleUseAnalyticSolver]=true` is applicable to satellite systems in the very simple disk component. In particular, the following conditions must be met for it to be valid:

- The hot halo component must be of type `verySimple` or `verySimpleDelayed`;
- The satellite component must be of type `verySimple`;
- The disk component must be of type `verySimple`;
- Properties of all other components must not change for satellite galaxies (with the exception of the `time` property of the basic component), or else the component must be `null`;
- Disk star formation and outflow rates must scale as $M_{\rm gas}/\tau$ where $M_{\rm gas}$ is the instantaneous mass of gas in the galaxy disk, and τ is a fixed timescale for any given satellite galaxy.

Under these conditions, the flow of mass between gas, stellar, and outflowed phases is analytically solvable.

12.2. Accretion of Gas into Halos

The accretion rate of gas from the IGM into a dark matter halo is expected to depend on (at least) the rate at which that halo mass is growing, the depth of its potential well and the thermodynamical properties of the accreting gas. Galacticus implements the following calculations of gas accretion from the IGM, which can be selected via the `accretionHaloMethod` input parameter.

12.2.1. Simple

Selected using `accretionHaloMethod=simple`, this method sets the accretion rate of baryons into a halo to be:

$$\dot{M}_{\rm accretion} = \begin{cases} (\Omega_{\rm b}/\Omega_{\rm M}) \dot{M}_{\rm halo} & \text{if } V_{\rm virial} > V_{\rm reionization} \text{ or } z > z_{\rm reionization} \\ 0 & \text{otherwise,} \end{cases} \tag{12.1}$$

where $z_{\rm reionization}$ = `[reionizationSuppressionRedshift]` is the redshift at which the Universe is reionized and $V_{\rm reionization}$ = `[reionizationSuppressionVelocity]` is the virial velocity below which accretion is suppressed after reionization. Setting $V_{\rm reionization}$ to zero will effectively switch off the effects of reionization on the accretion of baryonics. This algorithm attempts to offer a simple prescription for the effects of reionization and has been explored by multiple authors (e.g. Benson et al. 2002). In particular, Font

et al. [2010] show that it produces results in good agreement with more elaborate treatments of reionization. For halos below the accretion threshold, any accretion rate that would have otherwise occurred is instead placed into the "failed" accretion rate. For halos which can accrete, and which have some mass in their "failed" reservoir, that mass will be added to the regular accretion rate at a rate equal to the mass of the "failed" reservoir times the specific growth rate of the halo. The gas accreted is assumed to be from a pristine IGM and so has zero abundances. Chemical abundances are computed from the chemical state functions (see §12.30).

Note that, if $\dot{M}_{\rm halo} < 0$ then negative accretion rates of gas into the node can result. This ccan be prevented by setting `[accretionHalosSimpleNegativeAccretionAllowed]`=`false`.

By default, gas is accreted whenever the halo is growing in total mass. However, setting `[accretionHalosSimpleAccreteNe` causes accretion to occur only if the node mass is growing and exceeds the previous maximum node mass achieved along this branch of the merger tree. This requires use of a basic component which tracks the maximum mass along the branch (i.e. the `massMaximum` property).

12.2.2. Null

Selected using `accretionHaloMethod`=`null`, this method sets the accretion rate of baryons into a halo to be zero always.

12.2.3. Cold Mode

Selected using `accretionHaloMethod`=`coldMode`, this class extends the `simple` class by dividing the accretion into hot and cold mode components. The cold mode fraction follows the approximation introduced by Benson and Bower [2010], namely:

$$f_{\rm cold} = (1 + r^{1/\delta})^{-1}, \tag{12.2}$$

where $\delta =$ `[accretionColdModeShockStabilityTransitionWidth]`, $r = \epsilon_{\rm crit}/\epsilon$, and

$$\epsilon = r_{\rm s}\Lambda/\rho_{\rm s}v_{\rm s}^3, \tag{12.3}$$

where $r_{\rm s}$ is the accretion shock radius, Λ is the post-shock cooling function, $\rho_{\rm s}$ is the pre-shock density, $v_{\rm s}$ is the pre-shock velocity, and $\epsilon_{\rm crit} =$ `[accretionColdModeShockStabilityThreshold]`. The pre-shock radius is set equal to the halo virial radius, the pre-shock velocity is set equal to the halo virial velocity, while the pre-shock density is given by

$$\rho_{\rm s} = \frac{\gamma-1}{\gamma+1}\frac{3}{4\pi}\frac{\Omega_{\rm b}}{\Omega_{\rm m}}\frac{M}{r_{\rm s}^3}\left[1+\frac{(\alpha+3)(10+9\pi)}{4}\right]^{-1}, \tag{12.4}$$

where M is the total halo mass, $\gamma(=5/3)$ is the adiabatic index of the gas, and α is the exponent of the initial power-law density perturbation ($\alpha = 0$ is assumed). The post-shock density and temperature are found assuming the strong-shock limit.

12.2.4. Naoz & Barkana (2007)

Selected using `accretionHaloMethod`=`naozBarkana2007` this method uses the filtering mass prescription of Naoz and Barkana [2007] to compute the accreted rate of baryonic material. Specifically, Naoz and Barkana [2007] assume that the gas mass content of halos is given by $M_{\rm g}(M_{\rm 200b}, M_{\rm F}) = (\Omega_{\rm b}/\Omega_{\rm M})f(M_{\rm 200b}/M_{\rm F})M_{\rm 200b}$ where $M_{\rm F}$ is the filtering mass, as first introduced by Gnedin [2000] but defined following Naoz and Barkana [2007], $M_{\rm 200b}$ is the halo mass defined by a density threshold of 200 times the mean background, and

$$f(x) = [1 - (2^{1/3} - 1)x^{-1}]^{-3}. \tag{12.5}$$

The accretion rate onto the halo is therefore assumed to be

$$\dot{M}_\mathrm{g} = \frac{\Omega_\mathrm{b}}{\Omega_\mathrm{M}} \frac{\mathrm{d}}{\mathrm{d}M_\mathrm{200b}} \left[f(M_\mathrm{200b}/M_\mathrm{F}) M_\mathrm{200b} \right] \dot{M}_\mathrm{total}. \tag{12.6}$$

This would result in a precise match to the Naoz and Barkana [2007] assumption if:

1. The filtering mass is constant in time;
2. M_total corresponds to M_200b; and
3. The growth of halos occurs through smooth accretion, not through merging of smaller halos.

In practice all three assumptions are violated. As such, the mass fraction in the halo will differ from $f(M_\mathrm{200b}/M_\mathrm{F})$. To address this issue, mass is additionally assumed to flow from the hot halo reservoir to the unaccreted mass reservoir at a rate:

$$\dot{M}_\mathrm{hot} = -\frac{\alpha_\mathrm{adjust}}{\tau_\mathrm{dyn}} [M_\mathrm{hot} + M_\mathrm{unaccreted}][f_\mathrm{accreted} - f(M_\mathrm{200b}/M_\mathrm{F})], \tag{12.7}$$

where α_adjust =[`accretionHaloNaozBarkana2007RateAdjust`].

12.3. Accretion of Total Mass into Halos

The accretion rate of total mass (i.e. the mass which would be accreted in a dark matter-only universe) can be computed by the following options, which can be selected via the `accretionHaloTotalMethod` input parameter.

12.3.1. Simple

Selected using `accretionHaloTotalMethod=simple`, this class assumes that the accretion rate is equal to the growth rate of the basic component mass property.

12.3.2. Bertschinger

Selected using `accretionHaloTotalMethod=bertschinger`, this class assumes that the accretion rate is equal to the growth rate of the basic component Bertschinger mass property.

12.4. Cosmology Functions

The background cosmology describes the evolution of an isotropic, homoegeneous Universe within which our calculations are carried out. For the purposes of Galacticus, the background cosmology is used to relate expansion factor/redshift to cosmic time and to compute the density of various components (e.g. dark matter, dark energy, etc.) at different epochs. Background cosmological models are specified via the `cosmologyMethod`, and the physics that must be implemented for each cosmological model is describe in more detail in §16.4.1. Currently implemented cosmological models are as follows.

12.4.1. Matter + Lambda

Selected with `cosmologyMethod=matter-lambda`, in this implementation cosmological relations are computed assuming a universe that contains only collisionless matter and a cosmological constant.

12.4.2. Matter + Dark Energy

Selected with `cosmologyMethod=matter-darkEnergy`, in this implementation cosmological relations are computed assuming a universe that contains only collisionless matter and dark energy with an equation of state $w(a) = w_0 + w_1 a(1-a)$ [Jassal et al., 2005], with $w_0 =$`[darkEnergyEquationOfStateW0]`, and $w_1 =$`[darkEnergyEquationOfStateW1]`.

12.5. Circumnuclear Accretion Disks

Circumnuclear accretion disks surrounding supermassive black holes at the centers of galaxies influence the evolution of both the black hole (via accretion rates of mass and angular momentum and possibly by extracting rotational energy from the black hole) and the surrounding galaxy if they lead to energetic outflows (e.g. jets) from the nuclear region. Accretion disk type is specified via the `accretionDisksMethod`, and the physics that must be implemented for each accretion disk type is describe in more detail in §16.4.1. Current implementations of accretion disks are as follows.

12.5.1. Shakura-Sunyaev Geometrically Thin, Radiatively Efficient Disks

Selected with `accretionDisksMethod=Shakura-Sunyaev`, this implementation assumes that accretion disks are always described by a radiatively efficient, geometrically thin accretion disk as described by Shakura and Sunyaev [1973]. The radiative efficiency of the flow is computed assuming that material falls into the black hole without further energy loss from the innermost stable circular orbit (ISCO), while the spin-up rate of the black hole is computed assuming that the material enters the black hole with the specific angular momentum of the ISCO (i.e. there are no torques on the material once it begins to fall in from the ISCO; Bardeen 1970). For these thin disks, jet power is computed, using the expressions from Meier (2001; his equations 4 and 5).

12.5.2. Advection Dominated, Geometrically Thick, Radiatively Inefficient Flows (ADAFs)

Selected with `accretionDisksMethod=ADAF`, this implementation assumes that accretion is via an advection-dominated accretion flow (ADAF) [Narayan and Yi, 1994] which is radiatively inefficient and geometrically thick. The radiative efficiency of the flow, which will be zero for a pure ADAF, is controlled by `[adafRadiativeEfficiencyType]`. If set to `fixed`, then the radiative efficiency is set to the value of the input parameter `[adafRadiativeEfficiency]`. Alternatively, if set to `thinDisk` the radiative efficiency will be set to that of a Shakura-Sunyaev thin disk. The spin up rate of the black hole and the jet power produced as material accretes into the black hole are computed using the method of Benson and Babul [2009]. The maximum efficiency of the jet (in units of the accretion power $\dot{M}c^2$) is set by `[adafJetEfficiencyMaximum]`—in the model of Benson and Babul [2009] the jet efficiency diverges as $j \rightarrow 1$, setting a maximum is important to avoid numerical instabilities. The energy of the accreted material can be set equal to the energy at infinity (as expected for a pure ADAF) or the energy at the ISCO by use of the `[adafEnergyOption]` parameter (set to `pureADAF` or `ISCO` respectively). The ADAF structure is controlled by the adiabatic index, γ, and viscosity parameter, α, which are specified via the `[adafAdiabaticIndex]` and `[adafViscosityOption]` input parameters respectively. The field-enhancing shear, g, is computed using $g = \exp(\omega\tau)$ if `[adafFieldEnhanceType]` is set to "exponential" where ω is the frame-dragging frequency and τ is the smaller of the radial inflow and azimuthal velocity timescales. If `[adafFieldEnhanceType]` is set to "linear" then the alternative version, $g = 1 + \omega\tau$ is used instead. `[adafViscosityOption]` may be set to "`fit`", in which case the fitting function for α as a

function of black hole spin is used:

$$\alpha(j) = 0.015 + 0.02j^4 \quad \text{if} \quad g = \exp(\omega\tau) \text{ and } E = E_{\rm ISCO}, \tag{12.8}$$
$$\alpha(j) = 0.025 + 0.08j^4 \quad \text{if} \quad g = 1 + \omega\tau \text{ and } E = E_{\rm ISCO}, \tag{12.9}$$
$$\alpha(j) = 0.010 + 0.00j^4 \quad \text{if} \quad g = \exp(\omega\tau) \text{ and } E = 1, \tag{12.10}$$
$$\alpha(j) = 0.025 + 0.02j^4 \quad \text{if} \quad g = 1 + \omega\tau \text{ and } E = 1. \tag{12.11}$$

12.5.3. "Switched" Disks

Selected with `accretionDisksMethod=switched`, this method allows for accretion disks to switched between radiatively efficient (Shakura-Sunyaev) and inefficient (ADAF) modes. This is intended to crudely model the fact that accretion disks are expected to be radiatively inefficient at both high accretion rates (i.e. as they approach the Eddington luminosity the radiation pressure from a radiatively efficient flow would begin to disrupt the flow itself by definition), while accretion flows at low accretion rates ($\dot{M}_{\rm acc} < \alpha^2 \dot{M}_{\rm Edd}$, where $\alpha \sim 0.1$ is the usual parameter controlling the rate of angular momentum transport in a Shakura and Sunyaev [1973] accretion disk) are also radiatively inefficient as radiative processes are too inefficient at the associated low densities to radiate energy at the rate it is being liberated. A more thorough discussion is given by Begelman [2014].

The properties of the switched disk (e.g. radiative efficiency, jet power), are a linear combination of those of the Shakura-Sunyaev and ADAF modes, with the ADAF fraction being given by:

$$f_{\rm ADAF} = [1 + \exp(y_{\rm min})]^{-1} + [1 + \exp(y_{\rm max})]^{-1}, \tag{12.12}$$

where

$$y_{\rm min} = +\log(x/x_{\rm min})/\Delta_x, \tag{12.13}$$
$$y_{\rm max} = -\log(x/x_{\rm max})/\Delta_x, \tag{12.14}$$

and,

$$x = \dot{M}/\dot{M}_{\rm Eddington}. \tag{12.15}$$

Here, $x_{\rm min}$ =`[accretionRateThinDiskMinimum]`, $x_{\rm max}$ =`[accretionRateThinDiskMaximum]`, and Δ_x=`[accretionRateTra` If either `[accretionRateThinDiskMinimum]` or `[accretionRateThinDiskMaximum]` is set to "`none`" then the corresponding term in eqn. (12.12) is excluded.

Additionally, if `[accretionDiskSwitchedScaleAdafRadiativeEfficiency]` is set to `true` then the radiative efficiency of the ADAF component is reduced by a factor $x/x_{\rm min}$ when $x < x_{\rm min}$.

12.5.4. Eddington-limited Disks

Selected with `accretionDisksMethod=eddingtonLimited`, this method does not assume any physical model for the accretion disk, but merely assumes that jets are powered at a fixed fraction `[accretionDiskJetPowerEddington` of the Eddington luminosity. The radiative efficiency is similarly set at a fixed value of `[accretionDiskRadiativeEfficiency` Since no physical model for the disk is assumed, the black hole spin up rate is always set to zero.

12.6. Dark Matter Structure Formation

A variety of functions are used to describe structure formation in dark matter dominated universes. These are described below.

12.6.1. Primordial Power Spectrum

The functional form of the primordial dark matter power spectrum is selected via the `powerSpectrumMethod` parameter. The power spectrum is computed from the specified primordial power spectrum and the transfer function (see §16.4.1) and normalized to a value of σ_8 specified by `[sigma_8]`.

(Running) Power Law Spectrum

Selected via `powerSpectrumMethod=powerLaw`, this method implements a primordial power spectrum of the form:

$$P(k) \propto k^{n_{\rm eff}(k)}, \tag{12.16}$$

where

$$n_{\rm eff}(k) = n_{\rm s} + \frac{1}{2}\frac{{\rm d}n}{{\rm d}\ln k}\ln\left(\frac{k}{k_{\rm ref}}\right), \tag{12.17}$$

where $n_{\rm s}$ =`[index]` is the power spectrum index at wavenumber $k_{\rm ref}$ =`[wavenumberReference]` and ${\rm d}n/{\rm d}\ln k$ =`[running]` describes the running of this index with wavenumber.

12.6.2. Cosmological Mass Root Variance

The method used to compute the root variance of the cosmological mass field, $\sigma(M)$, is selected via the `cosmologicalMassVarianceMethod` parameter.

Filtered Power Spectrum

Selected via `cosmologicalMassVarianceMethod=filteredPowerSpectrum`, this method computes the mass root variance using:

$$\sigma^2(M) = \frac{1}{2\pi^2}\int_0^\infty P(k)T^2(k)W^2(k)k^2{\rm d}k \tag{12.18}$$

where $P(k)$ is the primordial power spectrum (see §12.6.1), $T(k)$ is the transfer function (see §12.6.5), and $W(k)$ is the power spectrum variance window function (see §12.6.3).

12.6.3. Power Spectrum Variance Window Function

The functional form of the window function used in computing the variance of the power spectrum is selected via the `powerSpectrumWindowFunctionMethod` parameter. Note that when computing the normalization of the power spectrum to match the specified value of σ_8 a top-hat real-space window function is *always* used (as per the definition of σ_8).

Top-hat

Selected via `powerSpectrumWindowFunctionMethod=topHat`, this method implements a top-hat window function in real-space:

$$W(k) = \frac{3(\sin(x) - x\cos(x))}{x^3}, \tag{12.19}$$

where $x = kR$ and $R = (3M/4\pi\bar{\rho})^{1/3}$ for a smoothing scale M and mean matter density $\bar{\rho}$.

Sharp in k-space

Selected via `powerSpectrumWindowFunctionMethod=kSpaceSharp`, this method implements a top-hat window function in k-space:

$$W(k) = \begin{cases} 1 & \text{if } k < k_\mathrm{s} \\ 0 & \text{if } k > k_\mathrm{s}, \end{cases} \tag{12.20}$$

where if `[powerSpectrumWindowFunctionSharpKSpaceNormalization]=natural` then $k_\mathrm{s} = (6\Pi^2\bar{\rho}/M)^{1/3}$ for a smoothing scale M and mean matter density $\bar{\rho}$. Otherwise, `[powerSpectrumWindowFunctionSharpKSpaceNormalizatio` must be set to a numerical value, α, in which case $k_\mathrm{s} = \alpha/R_\mathrm{th}$ with $R_\mathrm{th} = 3M/4\pi\bar{\rho}$ for a smoothing scale M and mean matter density $\bar{\rho}$.

Hybrid of Top-hat and Sharp in k-space

Selected via `powerSpectrumWindowFunctionMethod=topHatKSpaceSharpHybrid`, this method implements a convolution of a top-hat window function and sharp k-space window function in k-space:

$$W(k) = W_\mathrm{th}(k)W_\mathrm{s}(k), \tag{12.21}$$

where

$$W(k) = \frac{3(\sin(x) - x\cos(x))}{x^3}, \tag{12.22}$$

where $x = kR_\mathrm{th}$, and

$$W_\mathrm{s}(k) = \begin{cases} 1 & \text{if } k < k_\mathrm{s} \\ 0 & \text{if } k > k_\mathrm{s}, \end{cases} \tag{12.23}$$

where $ks = \alpha/R_\mathrm{s}$ if `[powerSpectrumWindowFunctionSharpKSpaceNormalization]` is assigned a numerical value. Alternatively, if `[powerSpectrumWindowFunctionSharpKSpaceNormalization]=natural` then the value of α is chosen such that $k_\mathrm{s} = (6\Pi^2\bar{\rho}/M)^{1/3}$ if $R_\mathrm{s} = 3M/4\pi\bar{\rho}$.

The radii, R_th and R_s, are chosen such that:

$$R_\mathrm{th}^2 + R_\mathrm{s}^2 = (3M/4\pi\bar{\rho})^{2/3} \tag{12.24}$$

$$R_\mathrm{s} = \beta R_\mathrm{th}, \tag{12.25}$$

where $\beta =$`[powerSpectrumWindowFunctionSharpKSpaceTopHatRadiiRatio]`.

12.6.4. Non-linear Matter Power Spectrum

The non-linear matter power spectrum method is selected via the `powerSpectrumNonlinearMethod` parameter.

Linear

Selected via `powerSpectrumNonlinearMethod=linear`, this method simply returns the linear matter power spectrum. It is intended primarily for testing purposes.

Peacock & Dodds (1996)

Selected via `powerSpectrumNonlinearMethod=peacockDodds1996`, this method usese the fitting function of Peacock and Dodds [1996] to compute the non-linear matter power spectrum.

CosmicEmu

Selected via `powerSpectrumNonlinearMethod=cosmicEmu`, this method usese the cosmic emulator ("CosmicEmu") code of Lawrence et al. [2010] to evaluate the non-linear matter power spectrum. The CosmicEmu code will be downloaded, compiled and run as necessary if this option is utilized.

12.6.5. Transfer Function

The functional form of the cold dark matter transfer function is selected via the `transferFunctionMethod` parameter. The power spectrum is computed from the specified transfer function and the primordial power spectrum (see §12.6.1) and normalized to a value of σ_8 specified by `[sigma_8]`.

Null

Selected with `transferFunctionMethod=null`, this method assumes $T(k) = 1$ for all k.

BBKS

Selected with `transferFunctionMethod=BBKS`, this method uses the fitting function of Bardeen et al. [1986] to compute the CDM transfer function. The BBKS warm dark matter transfer function can be used by specifying the appropriate streaming length (in Mpc) via the `[transferFunctionWDMFreeStreamingLength]` parameter.

Eisenstein & Hu

Selected with `transferFunctionMethod=Eisenstein-Hu1999`, this method uses the fitting function of Eisenstein and Hu [1999] to compute the CDM transfer function. It requires that the effective number of neutrino species be specified via the `effectiveNumberNeutrinos` parameter and summed mass of all neutrino species (in eV) be specified via the `summedNeutrinoMasses` parameter. Additionally, the transfer function can be modified to model warm dark matter using the fitting function given by Barkana et al. [2001]:

$$T(k) \rightarrow T(k)(1 + [\epsilon k R_{\rm c}^0]^{2\nu})^{-\eta/\nu}, \tag{12.26}$$

where $R_{\rm c}^0 =$`[transferFunctionWdmCutOffScale]`, $\epsilon =$`[transferFunctionWdmEpsilon]`, $\eta =$`[transferFunctionWdmEta]`, $\nu =$`[transferFunctionWdmNu]`.

CAMB

Selected with `transferFunctionMethod=CAMB`, this method uses the CAMB code to compute the CDM transfer function. It requires that the mass fraction of helium in the early Universe be specified via the `Y_He` parameter. CAMB will be downloaded and run if the transfer function needs to be computed. It will then be stored in a file for future reference.

File

Selected with `transferFunctionMethod=file`, this method reads a tabulated transfer function from an XML file (specified via the `transferFunctionFile` parameter), interpolating between tabulated points. The structure of the transfer function file is described in §16.4.1.

12.6.6. Linear Growth Function

The function describing the amplitude of linear perturbations is selected via the `linearGrowthMethod` parameter.

Simple

Selected with `linearGrowthMethod=simple`, this method calculates the growth of linear perturbations using standard perturbation theory in a Universe consisting of matter and a cosmological constant. Perturbations in the baryons are treated just as for dark matter (i.e. pressure forces are ignored), while perturbations in the radiation are assumed not to grow.

12.6.7. Critical Overdensity

The method used to compute the critical linear overdensity at which overdense regions virialize is selected via the `criticalOverdensityMethod` parameter.

Spherical Collapse (Matter + Cosmological Constant)

Selected with `criticalOverdensityMethod=sphericalTopHat` this method calculates critical overdensity using a spherical top-hat collapse model assuming a Universe which contains matter and a cosmological constant (see, for example, Percival 2005).

Kitayama & Suto (1996)

Selected with `criticalOverdensityMethod=Kitayama-Suto1996` this method calculates critical overdensity using the fitting formula of Kitayama and Suto [1996], and so is valid only in flat cosmological models (an error will be reported in non-flat models). Specifically,

$$\delta_{\rm crit}(t) = \frac{3(12\pi)^{2/3}}{20}[1 + 0.0123 \log_{10}\{\Omega_{\rm matter}(t)\}]/D(t). \tag{12.27}$$

Fixed

Selected with `criticalOverdensityMethod=fixed` this method sets the critical overdensity equal to a fixed value (set via the parameter `[criticalOverdensityFixed]`) divided by the linear growth factor.

12.6.8. Critical Overdensity Mass Scaling

The method used to compute the scaling with mass of the critical linear overdensity at which overdense regions virialize is selected via the `criticalOverdensityMassScalingMethod` parameter.

Null

Selected with `criticalOverdensityMassScalingMethod=null` this method assumes that the critical overdensity is independent of mass.

Warm Dark Matter

Selected with `criticalOverdensityMassScalingMethod=warmDarkMatter` this method assumes that the critical overdensity scales with mass as expected for warm dark matter using the results of Barkana et al. [2001]. Specifically, the critical overdensity is multiplied by a factor

$$\exp\left[\left(\frac{M_{\rm J}}{8M}\right)^{1.40} + \left(\frac{M_{\rm J}}{8M}\right)^{0.45}\right], \tag{12.28}$$

where M is the mass in question, M_J is the effective Jeans mass of the warm dark matter as defined by Barkana et al. [2001; their eqn. 10]:

$$M_\mathrm{J} = 3.06 \times 10^8 \left(\frac{1+z_\mathrm{eq}}{3000}\right)^{1.5} \left(\frac{\Omega_\mathrm{M} h_0^2}{0.15}\right)^{1/2} \left(\frac{g_\mathrm{X}}{1.5}\right)^{-1} (m_\mathrm{X}/1.0\mathrm{keV})^{-4}, \tag{12.29}$$

the redshift of matter-radiation equality is given by

$$z_\mathrm{eq} = 3600 \left(\frac{\Omega_\mathrm{M} h_0^2}{0.15}\right) - 1, \tag{12.30}$$

and g_X and m_X are the effective number of degrees of freedom and the mass of the warm dark matter particle respectively. This fitting function has been found the fit the numerical results of Barkana et al. [2001] well.

12.6.9. Virial Density Contrast

The method used to compute the mean density contrast of virialized dark matter halos is selected via the `virialDensityContrastMethod` parameter.

Bryan & Norman (1998) Fitting Function

Selected with `virialDensityContrastMethod=Bryan-Norman1998` this method calculates virial density contrast using the fitting functions given by Bryan and Norman [1998]. As such, it is valid only for $\Omega_\Lambda = 0$ or $\Omega_\mathrm{M} + \Omega_\Lambda = 1$ cosmologies and will abort on other cosmologies.

Fixed

Selected with `virialDensityContrastMethod=fixed` this method uses a fixed virial density contrast of `[virialDensityConstrastFixed]`, defined relative to `criticalDensity` and `meanDensity` as specified by `[virialDensityConstrastFixedType]`.

Spherical Collapse (Matter + Cosmological Constant)

Selected with `virialDensityContrastMethod=sphericalTopHat` this method calculates virial density contrast using a spherical top-hat collapse model assuming a Universe which contains matter and a cosmological constant (see, for example, Percival 2005).

Kitayama & Suto (1996)

Selected with `virialDensityContrastMethod=Kitayama-Suto1996` this method calculates virial density contrast using the fitting formula of Kitayama and Suto [1996], and so is valid only in flat cosmological models (an error will be reported in non-flat models). Specifically,

$$\Delta_\mathrm{virial}(t) = 18\pi^2[1 + 0.4093 \left\{\frac{1}{\Omega_\mathrm{matter}(t)} - 1\right\}(t)^{0.9052}]. \tag{12.31}$$

Friends-of-Friends

Selected with `virialDensityContrastMethod=friendsOfFriends` this method computes the virial density contrast consistent with a given friends-of-friends algorithm linking length. According to Lacey and

Cole [1994], the friends-of-friends algorithm selects objects bounded by an isodensity contour of density contrast

$$\Delta_{\rm iso} = \frac{3}{2\pi b^3}, \tag{12.32}$$

where $b =$ `[virialDensityContrastFoFLinkingLength]` is the dimensionless linking length of the algorithm (i.e. the linking length in units of the mean interparticle spacing). The virial density contrast is then given by:

$$\Delta_{\rm vir} = \frac{\bar{\rho}_{\rm vir}}{\rho(r_{\rm vir})}\Delta_{\rm iso}, \tag{12.33}$$

where $\bar{\rho}_{\rm vir}$ is the mean density inside the virial radius and $\rho(r_{\rm vir})$ is the density at the virial radius. The ratio $\bar{\rho}_{\rm vir}/\rho(r_{\rm vir})$ is specified via the parameter `[virialDensityContrastFoFDensityRatio]`. Its default value of 4.688 is appropriate for an NFW halo of concentration $c = 6.88$ which is the concentration found by Prada et al. [2011] for halos with $\sigma = 1.686$ which is the approximate critical overdensity for collapse).

Percolation

Selected with `virialDensityContrastMethod=percolation` this method computes the virial density contrast consistent with a given friends-of-friends algorithm linking length using the percolation-theory-motivated calibration of More et al. [2011]. Specifically, the friends-of-friends algorithm is assumed to link together particles forming an isodensity surface of density $\rho = \bar{\rho} n_{\rm c}/b^3$, where $\bar{\rho}$ is the mean density of the universe, $n_{\rm c} = 0.652960$ is a critical density for percolation as given by More et al. [2011], and b is the linking length. Given this bounding density, the virial density contrast is found by requiring that the halo contain the required mass within such a bounding density, given the halo density profile.

12.6.10. Halo Bias

The dark matter halo linear bias method is selected via the `darkMatterHaloBiasMethod` parameter.

Press-Schechter

Selected with `darkMatterHaloBiasMethod=Press-Schechter` this method uses a bias consistent with the halo mass function of Press and Schechter [1974] (see [Mo and White, 1996]).

Sheth-Tormen

Selected with `darkMatterHaloBiasMethod=SMT` this method uses a bias consistent with the halo mass function of Sheth et al. [2001].

Tinker

Selected with `darkMatterHaloBiasMethod=Tinker2010` this method uses the functional form proposed by Tinker et al. [2010] to compute the halo bias. The bias is computed at the appropriate virial overdensity (see §12.6.9).

12.6.11. Halo Mass Function

The dark matter halo mass function (i.e. the number of halos per unit volume per unit mass interval) is selected via the `haloMassFunctionMethod` parameter.

Press-Schechter

Selected with `haloMassFunctionMethod=pressSchechter` this method uses the functional form proposed by Press and Schechter [1974] to compute the halo mass function. Specifically,

$$n(M,t) = 2\frac{\Omega_{\rm M}\rho_{\rm crit}}{M^2}\alpha\sigma^2(M)f[S(M,t)], \tag{12.34}$$

where $\alpha = \mathrm{d}\ln\sigma/\mathrm{d}\ln M$ and $f[S]$ is the excursion set barrier first crossing distribution for variance $S(M) = \sigma^2(M)$, computed using the selected `excursionSetFirstCrossingMethod` (see §16.4.1).

Sheth-Tormen

Selected with `haloMassFunctionMethod=shethTormen` this method uses the functional form proposed by Sheth et al. [2001] to compute the halo mass function.

Despali et al. (2015)

Selected with `haloMassFunctionMethod=despali2015` this method uses the functional form proposed by [Sheth et al., 2001, see §12.6.11] but with parameters a, p, and A set using eqn. (12) of Despali et al. [2015].

Tinker et al. (2008)

Selected with `haloMassFunctionMethod=tinker2008` this method uses the functional form proposed by Tinker et al. [2008] to compute the halo mass function. The mass function is computed at the appropriate virial overdensity (see §12.6.9).

Simple Systematic

Selected with `haloMassFunctionMethod=simpleSystematic` this method modifies another mass function (specified as a subparameter) as follows:

$$\frac{\mathrm{d}n}{\mathrm{d}M}(M) \rightarrow \frac{\mathrm{d}n}{\mathrm{d}M}(M)\left(1+\alpha+\beta\log_{10}\left[\frac{M}{10^{12}M_\odot}\right]\right) \tag{12.35}$$

where $\alpha =$`[alpha]`, and $\beta =$`[beta]`.

12.7. Cold Mode Infall Ratres

The method to be used for computing the rate of infall of gas from the cold mode is specified by the `coldModeInfallRateMethod` parameter.

12.7.1. Dynamical Time

The infall rate from the cold mode is given by

$$\dot{M}_{\rm infall,coldmode} = f_{\rm infall,coldmode}\frac{M_{\rm coldmode}}{\tau_{\rm dyn}}, \tag{12.36}$$

where $f_{\rm infall,coldmode} =$`[dynamicalRateFraction]`.

12.8. Conditional Mass Functions

The method to be used for computing the conditional mass function (for use in data modelling) is specified by the `conditionalMassFunctionMethod` parameter.

12.8.1. Behroozi et al. (2010)

Selected with `conditionalMassFunctionMethod=Behroozi2010`, this method uses the functional form given by Behroozi et al. (2010; see also Leauthaud et al. 2011). The mean number of central galaxies of stellar mass, $M_\star$, in halos of mass M_h is given by:

$$\log_{10}[f^{-1}_\mathrm{SHMR}(M_\star)] = \log_{10}(M_\mathrm{h}) = \log_{10}(M_1) + \beta \log_{10}\left(\frac{M_\star}{M_{\star,0}}\right) + \frac{(M_\star/M_{\star,0})^\delta}{1+(M_\star/M_{\star,0})^{-\gamma}} - \frac{1}{2}, \tag{12.37}$$

where

$$\langle n_\mathrm{cen}(M_\mathrm{h}|M^\rangle_\star\rangle = \frac{1}{2}\left[1-\mathrm{erf}\left(\frac{\log_{10}(M_\star)-\log_{10}[f_\mathrm{SHMR}(M_\mathrm{h})]}{\sqrt{2}\sigma_{\log M_\star}}\right)\right]. \tag{12.38}$$

The mean number of satellite galaxies of stellar mass, $M_\star$, in halos of mass M_h is given by:

$$\langle N_\mathrm{sat}(M_\mathrm{h}|M_\star)\rangle = \langle n_\mathrm{cen}(M_\mathrm{h}|M^\rangle_\star\rangle \left(\frac{f^{-1}_\mathrm{SHMR}(M_\star)}{M_\mathrm{sat}}\right)^{\alpha_\mathrm{sat}} \exp\left(-\frac{M_\mathrm{cut}}{f^{-1}_\mathrm{SHMR}(M_\star)}\right), \tag{12.39}$$

where

$$\frac{M_\mathrm{sat}}{10^{12}M_\odot} = B_\mathrm{sat}\left(\frac{f^{-1}_\mathrm{SHMR}(M_\star)}{10^{12}M_\odot}\right)^{\beta_\mathrm{sat}}, \tag{12.40}$$

and

$$\frac{M_\mathrm{cut}}{10^{12}M_\odot} = B_\mathrm{cut}\left(\frac{f^{-1}_\mathrm{SHMR}(M_\star)}{10^{12}M_\odot}\right)^{\beta_\mathrm{cut}}. \tag{12.41}$$

The number of centrals in a given halo is assumed to be Bernoulli distributed (0 or 1), while the number of satellites is assumed to be Poisson distributed with the above mean. The parameters of the distribution are controlled by the input parameters as listed in Table 12.1.

12.9. Cooling of Gas Inside Halos

The cooling of gas within dark matter halos is controlled by a number of different algorithms which will be decribed below.

12.9.1. Cooling Function

The cooling function of gas, $\Lambda(\rho, T, \mathbf{Z})$, is implemented by the algorithm(s) selected using the `coolingFunctionMethods` parameter. If more than one cooling function is specified, then the net cooling function is a sum over all of those selected.

Atomic Collisional Ionization Equilibrium Using Cloudy

Selected using `coolingFunctionMethods=atomic_CIE_Cloudy`, this method computes the cooling function using the Cloudy code and under the assumption of collisional ionization equilibrium with no molecular contribution. Abundances are Solar, except for zero metallicity calculations which use Cloudy's "primordial" metallicity. The helium abundance for non-zero metallicity is scaled between primordial and

Table 12.1.: The correspondance between parameters of the Behroozi et al. [2010] model for the conditional mass function and GALACTICUS input parameters.

Model parameter	Input parameter
$\alpha_{\rm sat}$	`conditionalMassFunctionBehrooziAlphaSatellite`
$\log_{10}(M_1/M_\odot)$	`conditionalMassFunctionBehrooziLog10M1`
$\log_{10}(M_{\star,0}/M_\odot)$	`conditionalMassFunctionBehrooziLog10Mstar0`
β	`conditionalMassFunctionBehrooziBeta`
δ	`conditionalMassFunctionBehrooziDelta`
γ	`conditionalMassFunctionBehrooziGamma`
$\sigma_{\log M_\star}$	`conditionalMassFunctionBehrooziSigmaLogMstar`
$B_{\rm cut}$	`conditionalMassFunctionBehrooziBCut`
$B_{\rm sat}$	`conditionalMassFunctionBehrooziBSatellite`
$\beta_{\rm cut}$	`conditionalMassFunctionBehrooziBetaCut`
$\beta_{\rm sat}$	`conditionalMassFunctionBehrooziBetaSatellite`

Solar values linearly with metallicity. The CLOUDY code will be downloaded and run to compute the cooling function as needed, which will then be stored for future use. As this process is slow, a precomputed table is provided with GALACTICUS. If metallicities outside the range tabulated in this file are required it will be regenerated with an appropriate range.

Collisional Ionization Eqiulibrium From File

Selected using `coolingFunctionMethods=CIE_from_file`, in this method the cooling function is read from a file specified by the `coolingFunctionFile` parameter. The format of this file is specified in §16.4.1. The cooling function is assumed to be computed under conditions of collisional ionization equilibrium and therefore to scale as ρ^2.

CMB Compton Cooling

Selected using `coolingFunctionMethods=CMB_Compton`, this method computes the cooling function due to Compton scattering off of CMB photons:

$$\Lambda = \frac{4\sigma_{\rm T} a k_{\rm B} n_{\rm e}}{m_{\rm e} c} T_{\rm CMB}^4 \left(T - T_{\rm CMB}\right), \tag{12.42}$$

where $\sigma_{\rm T}$ is the Thompson cross-section, a is the radiation constant, $k_{\rm B}$ is Boltzmann's constant, $n_{\rm e}$ is the number density of electrons, $m_{\rm e}$ is the electron mass, c is the speed of light, $T_{\rm CMB}$ is the CMB temperature at the current cosmic epoch and T is the temperature of the gas. The electron density is computed from the selected chemical state method (see §12.30).

Molecular Hydrogen (Galli-Palla)

Selected using `coolingFunctionMethods=molecularHydrogenGalliPalla`, this method computes the cooling function due to molecular hydrogen using the results of Galli and Palla [1998]. For the H–H_2 cooling function, the fitting functions from Galli and Palla [1998] are used. For the H_2^+–e^- and H–H_2^+ cooling functions fitting functions to the results plotted in Suchkov and Shchekinov [1978] are used:

$$\log_{10}\left(\frac{\Lambda(T)}{\rm erg\ s^{-1} cm^3}\right) = C_0 + C_1 \log_{10}\left(\frac{T}{\rm K}\right) + C_2 \left[\log_{10}\left(\frac{T}{\rm K}\right)\right]^2, \tag{12.43}$$

Table 12.2.: Coefficients of H_2^+ cooling functions as appearing in the fitting function, eq. 12.43.

	Coefficient		
Interaction	C_0	C_1	C_2
H_2^+–e^-	-33.33	5.565	-0.4675
H–H_2^+	-35.28	5.862	-0.5124

where the coefficients C_{0-2} are given in Table 12.2.

12.9.2. Cooling Rate

The algorithm used to compute the rate at which gas drops out of the hot halo due to cooling is selected with the `coolingRateMethod` parameter.

Cole et al. (2000)

Selected with `coolingRateMethod=Cole2000`, this method computes the cooling rate using the algorithm of Cole et al. [2000]. The cooling rate is given by

$$\dot{M}_{\rm cool} = \begin{cases} 4\pi r_{\rm infall}^2 \rho(r_{\rm infall}) \dot{r}_{\rm infall} & \text{if } r_{\rm infall} < r_{\rm hot,outer} \\ 0 & \text{if } r_{\rm infall} \geq r_{\rm hot,outer}, \end{cases} \tag{12.44}$$

where $\rho(r)$ is the density profile of the hot halo, and $r_{\rm infall}$ is the infall radius (see §12.9.6). The cooling rate is also set to zero in halos with virial velocities below `[coolingCutOffVelocity]` at redshifts below `[coolingCutOffRedshift]`.

Simple

Selected with `coolingRateMethod=simple`, this method computes the cooling rate using

$$\dot{M}_{\rm cool} = M_{\rm hot}/\tau_{\rm cool}, \tag{12.45}$$

where $\tau_{\rm cool}$ = `[coolingRateSimpleTimescale]`.

Simple Scaling

Selected with `coolingRateMethod=simpleScaling`, this method computes the cooling rate using

$$\dot{M}_{\rm cool} = M_{\rm hot}/\tau_{\rm cool}(M_{\rm halo}, z), \tag{12.46}$$

where

$$\tau_{\rm cool} = \tau_{\rm cool,0}(1+z)^{\beta_{\rm cool}} \exp\left(\left[\frac{M_{\rm halo}}{M_{\rm transition}}\right]^{\gamma_{\rm cool}}\right), \tag{12.47}$$

$\tau_{\rm cool,0}$ = `[coolingRateSimpleScalingTimescale]`, $\beta_{\rm cool}$ = `[coolingRateSimpleScalingTimescaleExponent]`, $M_{\rm transition}$ = `[coolingRateSimpleScalingTransitionMass]`, and $\gamma_{\rm cool}$ = `[coolingRateSimpleScalingCutoffExponent]`.

Velocity Maximum Scaling

Selected with `coolingRateMethod=velocityMaximumScaling`, this method computes the cooling rate using

$$\dot{M}_{\rm cool} = M_{\rm hot}/\tau_{\rm cool}(V_{\rm max,halo}, z), \tag{12.48}$$

where

$$\tau_{\rm cool} = \max\left[\tau_{\rm infall}\left(\frac{V_{\rm max}}{200{\rm km/s}}\right)^{-\gamma_{\rm infall}}(1+z)^{\alpha_{\rm infall}}\left(1+\exp\left[\frac{\log_{10}(V_{\rm max}/(1+z)^{\delta_{\rm infall}}\mathcal{V}_{\rm infall})]}{\Delta\log_{10}\mathcal{V}_{\rm infall}}\right]\right)^{\beta_{\rm infall}}, \tau_{\rm infall,min}\right], \tag{12.49}$$

with $\tau_{\rm infall}$ = `[coolingRateVelocityMaximumScalingTimescale]`, $\tau_{\rm infall,min}$ = `[coolingRateVelocityMaximumScalingTimes`
$\alpha_{\rm infall}$ = `[coolingRateVelocityMaximumScalingTimescaleExponent]`, $\beta_{\rm infall}$ = `[coolingRateVelocityMaximumScalingCut`
$\gamma_{\rm infall}$ = `[coolingRateVelocityMaximumScalingVelocityExponent]`, $\delta_{\rm infall}$ = `[]coolingRateVelocityMaximumScalingCut[`
$\mathcal{V}_{\rm infall}$ = `[coolingRateVelocityMaximumScalingCutOffVelocity]`, and $\Delta\log_{10}\mathcal{V}_{\rm infall}$ = `[coolingRateVelocityMaximumScal`

White & Frenk

Selected with `coolingRateMethod=White-Frenk1991`, this method computes the cooling rate using the expression given by White and Frenk [1991], namely

$$\dot{M}_{\rm cool} = \begin{cases} 4\pi r_{\rm infall}^2 \rho(r_{\rm infall})\dot{r}_{\rm infall} & \text{if } r_{\rm infall} < r_{\rm hot,outer} \\ M_{\rm hot}/\tau_{\rm halo,dynamical} & \text{if } r_{\rm infall} \geq r_{\rm hot,outer}, \end{cases} \tag{12.50}$$

where $r_{\rm infall}$ is the infall radius (see §12.9.6) in the hot halo and $\rho(r)$ is the density profile of the hot halo.

12.9.3. Cooling Rate Modifier

Algorithms that modify the rate of cooling of gas are specified via the `coolingRateModifierMethod` directive.

Cut-Off

This method sets the cooling rate to zero in halos with virial velocities below `[coolingCutOffVelocity]` at redshifts below/above `[coolingCutOffRedshift]` for `[coolingCutOffWhen]=after/before`. In other halos the cooling rate is not modified.

12.9.4. Cooling Radius

The algorithm used to compute the cooling radius is selected via the `coolingRadiusMethod` parameter.

Simple

Selcted with `coolingRadiusMethod=simple`, this method computes the cooling radius by seeking the radius at which the time available for cooling (see §12.9.9) equals the cooling time (see §12.9.8). The growth rate is determined consistently based on the slope of the density profile, the density dependence of the cooling function and the rate at which the time available for cooling is increasing. This method assumes that the cooling time is a monotonic function of radius.

Isothermal

Selected with `coolingRadiusMethod=isothermal`, this method computes the cooling radius by assuming an isothermal density profile, and a cooling rate proportional to density squared. This implies a cooling time:

$$t_{\rm cool} \equiv \frac{E}{\dot{E}} \propto \rho(r)^{-1}. \tag{12.51}$$

The cooling radius is then derived using

$$\rho(r_{\rm cool}) \propto t_{\rm available}^{-1} \tag{12.52}$$

which implies

$$r_{\rm cool} = r_{\rm virial} \left(\frac{t_{\rm available}}{t_{\rm cool,virial}} \right)^{1/2}, \tag{12.53}$$

where $t_{\rm cool,virial}$ is the cooling time at the virial radius.

12.9.5. Cooling: Freefall Radius

The algorithm used to compute the freefall radius for cooling is selected via the `freefallRadiusMethod` parameter.

Dark Matter Halo

Selected with `freefallRadiusMethod=darkMatterHalo`, this method assumes that the freefall radius corresponds to the radius at which the freefall time in the dark matter halo equals the time available for freefall (see §12.9.10).

12.9.6. Cooling: Infall Radius

The algorithm used to compute the infall radius for cooling is selected via the `infallRadiusMethod` parameter.

Cooling Radius

Selected with `infallRadiusMethod=coolingRadius`, this method assumes that the infall radius equals the cooling radius (see §12.9.4).

Cooling and Freefall Radii

Selected with `infallRadiusMethod=cooling and freefall`, this method assumes that the infall radius is equal to the smaller of the cooling and freefall radii (see §12.9.4 and §12.9.5).

Dark Matter Halo

Selected with `freefallRadiusMethod=darkMatterHalo`, this method computes the freefall radius by finding the radius in the dark matter halo profile from which a test particle could have free-fallen to zero radius (assuming it began at rest) in the time available for freefall (see §12.9.10).

12.9.7. Cooling Specific Angular Momentum

The algorithm used to compute the specific angular momentum of cooling gas is selected via the `coolingSpecificAngularMome` parameter.

Constant Rotation

Selected with `coolingSpecificAngularMomentumMethod=constantRotation`, this implementation assumes that the specific angular momentum of cooling gas is given either by

$$j_{\rm cool} = \langle j \rangle r_{\rm cool} A, \tag{12.54}$$

where $r_{\rm cool}$ is the cooling radius, A is the rotation normalization (see) and $\langle j \rangle$ is the mean specific angular momentum of the cooling gas, if `[coolingAngularMomentumUseInteriorMean]=false`, or by

$$j_{\rm cool} = \langle j \rangle I_3(r_{\rm cool})/I_2(r_{\rm cool}) A, \tag{12.55}$$

where $I_n(r)$ is the $n^{\rm th}$ radial moment of the hot gas density profile from 0 to r (this therefore gives the mean specific angular momentum interior to radius r), if `[coolingAngularMomentumUseInteriorMean]=true`.

If `[coolingMeanAngularMomentumFrom]=darkMatter` then $\langle j \rangle$ is the mean specific angular momentum of the dark matter halo, computed from its spin parameter, while if `[coolingMeanAngularMomentumFrom]=hotGas` then $\langle j \rangle$ is equal to the mean specific angular momentum of gas currently in the hot gas reservoir. If `[coolingRotationVelocityFrom]=darkMatter` then the rotation normalization A is computed using the dark matter density profile, while if `[coolingRotationVelocityFrom]=hotGas` it is computed using the density profile of the hot gas reservoir.

Mean

Selcted with `coolingSpecificAngularMomentumMethod=mean`, this assumes that the specific angular momentum of cooling gas is given by

$$j_{\rm cool} = J_{\rm hot}/M_{\rm hot}, \tag{12.56}$$

where $J_{\rm hot}$ and $M_{\rm hot}$ are the total angular momentum and mass of the hot halo respectively.

12.9.8. Cooling Time

The algorithm used to compute the time taken for gas to cool (i.e. the cooling time) is selected via the `coolingTimeMethod` parameter.

Simple

Selcted with `coolingTimeMethod=simple`, this method assumes that the cooling time is simply

$$t_{\rm cool} = \frac{N}{2} \frac{{\rm k_B} T n_{\rm tot}}{\Lambda}, \tag{12.57}$$

where N =`degreesOfFreedom` is the number of degrees of freedom in the cooling gas which has temperature T and total particle number density (including electrons) $n_{\rm tot}$ and Λ is the cooling function.

12.9.9. Time Available for Cooling

The method used to determine the time available for cooling (i.e. the time for which gas in a halo has been able to cool) is selected by the `coolingTimeAvailableMethod` parameter.

Halo Formation

Selected with `coolingTimeAvailableMethod=haloFormation`, this methods assumes that the time available for cooling is equal to

$$t_{\rm available} = t - t_{\rm form}, \tag{12.58}$$

where $t_{\rm form}$ is the time at which the halo formed (see §11.16).

White & Frenk (1991)

Selected with `coolingTimeAvailableMethod=White-Frenk1991`, this methods assumes that the time available for cooling is equal to

$$t_{\rm available} = \exp\left[f \ln t_{\rm Universe} + (1-f) \ln t_{\rm dynamical}\right], \tag{12.59}$$

where f =`coolingTimeAvailableAgeFactor` is an interpolating factor, $t_{\rm Universe}$ is the age of the Universe and $t_{\rm dynamical}$ is the dynamical time in the halo. The original White and Frenk [1991] algorithm corresponds to $f = 1$.

12.9.10. Time Available for Freefall During Cooling

The method used to determine the time available for freefall during cooling calculations (i.e. the time for which gas in a halo has been able to freefall) is selected by the `freefallTimeAvailableMethod` parameter.

Halo Formation

Selected with `freefallTimeAvailableMethod=haloFormation`, this methods assumes that the time available for freefall is equal to

$$t_{\rm available} = t - t_{\rm form}, \tag{12.60}$$

where $t_{\rm form}$ is the time at which the halo formed (see §11.16).

12.10. Cosmology

The method used to compute cosmological relations (e.g. expansion factor as a function of time) is selected by the `cosmologyMethod` parameter.

12.10.1. Matter + Cosmological Constant Universes

Selected with `cosmologyMethod=matter-lambda`, this method assumes a universe which contains only matter and a cosmological constant

12.11. Dark Matter Halos

Several algorithms are used to implement dark matter halos.

12.11.1. Mass Accretion History

The method used to compute mass accretion histories of dark matter halos is selected via the `darkMatterHaloMassAccretionH` parameter.

Wechsler et al. (2002)

Selected with `darkMatterHaloMassAccretionHistoryMethod=wechsler2002`, under this method the mass accretion history is given by [Wechsler et al., 2002]:

$$M(t) = M(t_0) \exp\left(-2a_{\rm c}\left[\frac{a(t_0)}{a(t)} - 1\right]\right), \tag{12.61}$$

where t_0 is some reference time and a_c is a characteristic expansion factor defined by Wechsler et al. [2002] to correspond to the formation time of the halo (using the formation time definition of Bullock et al. 2001).

Zhao et al. (2009)

Selected with `darkMatterHaloMassAccretionHistoryMethod=zhao2009`, under this method the algorithm given by Zhao et al. [2009] to compute mass accretion histories. In particular, Zhao et al. [2009] give a fitting function for the quantity $\mathrm{d}\ln\sigma(M)/\mathrm{d}\ln\delta_c(t)$ for the dimensionless growth rate in a mass accretion history at time t and halo mass M. This is converted to a dimensionful growth rate using

$$\frac{\mathrm{d}M}{\mathrm{d}t} = \left(\frac{\mathrm{d}\ln\sigma(M)}{\mathrm{d}\ln M}\right)^{-1} \left(\frac{\mathrm{d}\delta_c(t)}{\mathrm{d}t}\right) \left(\frac{M}{\delta_c(t)}\right) \left(\frac{\mathrm{d}\ln\sigma(M)}{\mathrm{d}\ln\delta_c(t)}\right). \tag{12.62}$$

This differential equation is then solved numerically to find the mass accretion history.

12.11.2. Density Profile

The method uses to compute density profiles of dark matter halos is selected via the `darkMatterProfileMethod` parameter.

Isothermal

Selected with `darkMatterProfileMethod=isothermal`, under this method the density profile is given by:

$$\rho_{\mathrm{darkmatter}}(r) \propto r^{-2}, \tag{12.63}$$

normalized such that the total mass of the node is enclosed with the virial radius.

NFW

Selected with `darkMatterProfileMethod=NFW`, under this method the NFW density profile [Navarro et al., 1997] is used

$$\rho_{\mathrm{darkmatter}}(r) \propto \left(\frac{r}{r_\mathrm{s}}\right)^{-1} \left[1 + \left(\frac{r}{r_\mathrm{s}}\right)\right]^{-2}, \tag{12.64}$$

normalized such that the total mass of the node is enclosed with the virial radius and with the scale length $r_\mathrm{s} = r_\mathrm{virial}/c$ where c is the halo concentration (see §12.11.3).

Einasto

Selected with `darkMatterProfileMethod=Einasto`, under this method the Einasto density profile (e.g. Cardone et al. 2005) is used

$$\rho_{\mathrm{darkmatter}}(r) = \rho_{-2} \exp\left(-\frac{2}{\alpha}\left[\left(\frac{r}{r_{-2}}\right)^{\alpha} - 1\right]\right), \tag{12.65}$$

normalized such that the total mass of the node is enclosed with the virial radius and with the characteristic length $r_{-2} = r_\mathrm{virial}/c$ where c is the halo concentration (see §12.11.3). The shape parameter, α, is set using the density profile shape method (see §12.11.4).

Burkert

Selected with `darkMatterProfileMethod=Burkert`, under this method the NFW density profile [Burkert, 1995] is used

$$\rho_{\rm darkmatter}(r) \propto \left(1+\frac{r}{r_{\rm s}}\right)^{-1}\left(1+[\frac{r}{r_{\rm s}}]^2\right)^{-1}, \tag{12.66}$$

normalized such that the total mass of the node is enclosed with the virial radius and with the scale length $r_{\rm s} = r_{\rm virial}/c$ where c is the halo concentration (see §12.11.3). The mass enclosed within radius r is given by

$$M(<r) = M_{\rm virial}\frac{2\log(1+R)+\log(1+R^2)-2\tan^{-1}(R)}{2\log(1+c)+\log(1+c^2)-2\tan^{-1}(c)}, \tag{12.67}$$

where $R = r/r_{\rm s}$. The associated gravitational potential is

$$\Phi(r) = -{\rm G}\left(1+\frac{1}{R}\right)\frac{2\tan^{-1}(R)-2\log(1+R)+\log(1+R^2)}{-2\tan^{-1}(c)+2\log(1+c)+\log(1+c^2)}. \tag{12.68}$$

The peak of the rotation curve occurs at $R = 3.2446257246042642$ (found by numerical solution), and the Fourier transform of the profile, $F(k) = \int_0^c 4\pi r^2 \exp(-ikr)\rho(r){\rm d}r/kr$ (needed in calculations of clustering using the halo model) is given by

$$\begin{aligned} F(k) \quad = \quad & \{2\exp(-ik){\rm C_i}(k) - 2\exp(-ik){\rm C_i}(k[1+c]) + (1+i)\,[-i\exp(k)\pi - \exp(k){\rm E_i}(-k) \\ & + i\exp(-k){\rm E_i}(k) + \exp(k){\rm E_i}(i[i+c]k) - i\exp(-k){\rm E_i}(k[1+ic]) + (1+i)\exp(-ik){\rm S_i}(k) \\ & -(1+i)\exp(-ik){\rm S_i}(k[1+c])]\} \,/\, \left[k\left\{-2\tan^{-1}(c)+2\log(1+c)+\log(1+c^2)\right\}\right]. \end{aligned} \tag{12.69}$$

Tidally-heated

Selected with `darkMatterProfileMethod=tidallyHeated`, this method assumes that dark matter halos start out with a density profile defined by another `darkMatterProfileMethod` (specified via `[darkMatterProfileTidallyHea`
For subhalos, this profile is modified by tidal heating, under the assumption that the energy of a shell of mass before and after heating are related by

$$\frac{{\rm G}M'(r')}{r'} = \frac{{\rm G}M(r)}{r} + Qr^2, \tag{12.70}$$

where $M(r)$ is the mass enclosed within a radius r, and Q represents a normalized tidal heating (see §11.10.4 for example). Primes indicate values after heating, while unprimed variables indicate quantities prior to heating. With the assumption of no shell crossing, $M'(r') = M(r)$ and this equation can be solved for r given r'.

Not all methods are implemented for this profile. If `[darkMatterProfileTidallyHeatedUnimplementedIsFatal]=true` then attempts to call these methods in heated profiles aborts, otherwise, a warning is issued and the result for the unheated profile is used instead.

12.11.3. Dark Matter Halo Profile Concentration

The method uses to compute the concentrations of dark matter profiles is selected via the `darkMatterProfileConcentrationM` parameter.

Navarro, Frenk & White (1996)

Selected with `darkMatterProfileConcentrationMethod=NFW1996`, under this method the concentration is computed using the algorithm from Navarro et al. [1996]. In this algoritm, for a given halo of mass M at time t_0, a formation time is defined as the epoch at which there is a 50% probability (according to extended Press-Schechter theory) for a progenitor halo to have a mass greater than fM, where $f =$`[nfw96ConcentrationF]` is a parameter of the algorithm. This implies formation when the critical overdensity for collapse is

$$\delta_{\rm crit}(t_{\rm form}) = \left[2\nu_{1/2}^2 \left\{\sigma(fM)^2 2 - \sigma(M)^2\right\}\right]^{1/2} + \delta_{\rm crit}(t_0), \tag{12.71}$$

where $\nu_{1/2} = [{\rm erfc}^{-1}(1/2)]^{1/2}$. Navarro et al. [1996] then assume an overdensity at collapse of

$$\Delta(t_{\rm form}) = C\left[\frac{a(t_0)}{a(t_{\rm form})}\right]^3 \tag{12.72}$$

where $C =$`[nfw96ConcentrationC]` is a parameter of the algorithm. The concentration is then determined by solving

$$\frac{\Delta(t_{\rm form})}{\Delta_{\rm virial}(t_0)} = \frac{c^3}{3[\ln(1+c) - c/(1+c)]}. \tag{12.73}$$

Gao (2008)

Selected with `darkMatterProfileConcentrationMethod=gao2008`, under this method the concentration is computed using a fitting function from Gao et al. [2008]:

$$\log_{10} c = A \log_{10} M_{\rm halo} + B. \tag{12.74}$$

The parameters are a functon of expansion factor, a. We use the following fits to the Gao et al. [2008] results:

$$A = -0.140 \exp\left[-\left(\{\log_{10} a + 0.05\}/0.35\right)^2\right], \tag{12.75}$$

$$B = 2.646 \exp\left[-\left(\log_{10} a/0.50\right)^2\right]. \tag{12.76}$$

Zhao (2009)

Selected with `darkMatterProfileConcentrationMethod=zhao2009`, under this method the concentration is computed using a fitting function from Zhao et al. [2009]:

$$c = 4\left(1 + \left[\frac{t}{3.75 t_{\rm form}}\right]^{8.4}\right)^{1/8}, \tag{12.77}$$

where t is the time for the halo and $t_{\rm form}$ is a formation time defined by Zhao et al. [2009] as the time at which the main branch progenitor of the halo had a mass equal to 0.04 of the current halo mass. This formation time is computed directly from the merger tree branch associated with each halo. If the no branch exists or does not extend to the formation time then the formation time is computed by extrapolating the mass of the earliest resolved main branch progenitor to earlier times using the selected mass accretion history method (see §16.4.1).

Muñoz-Cuartas (2011)

Selected with `darkMatterProfileConcentrationMethod=munozCuartas2011`, under this method the concentration is computed using a fitting function from Muñoz-Cuartas et al. [2011]:

$$\log_{10} c = a \log_{10} \left(\frac{M_{\rm halo}}{h^{-1} M_\odot} \right) + b. \tag{12.78}$$

The parameters are a functon of redshift, z, given by

$$a = wz - m, \tag{12.79}$$

$$b = \frac{\alpha}{(z+\gamma)} + \frac{\beta}{(z+\gamma)^2}, \tag{12.80}$$

where $w = 0.029$, $m = 0.097$, $\alpha = -110.001$, $\beta = 2469.720$, $\gamma = 16.885$.

Prada et al. (2011)

Selected with `darkMatterProfileConcentrationMethod=prada2011`, under this method the concentration is computed using a fitting function from Prada et al. [2011]:

$$c(M,t) = B_0(x) \mathcal{C}(\sigma'), \tag{12.81}$$

where

$$\sigma'(M,t) = B_1(x)\sigma(M,t), \tag{12.82}$$

$$B_0(x) = c_{\rm min}(x)/c_{\rm min}(1.393), \tag{12.83}$$

$$B_1(x) = \sigma^{-1}_{\rm min}(x)/\sigma^{-1}_{\rm min}(1.393), \tag{12.84}$$

$$c_{\rm min}(x) = c_0 + (c_1 - c_0)[\tan^{-1}\{\alpha(x - x_0)\}/\Pi + 1/2], \tag{12.85}$$

$$\sigma^{-1}_{\rm min}(x) = \sigma_0^{-1} + (\sigma_1^{-1} - \sigma_0^{-1})[\tan^{-1}\{\beta(x - x_1)\}/\Pi + 1/2], \tag{12.86}$$

$$\mathcal{C}(\sigma') = A[(\sigma')/b)^c + 1] \exp(d/\sigma'^2), \tag{12.87}$$

$$x = (\Omega_\Lambda/\Omega_{\rm M})^{1/3} a(t), \tag{12.88}$$

with the following parameters (default values taken from Prada et al. [2011] given in []): $A =$ `[prada2011ConcentrationA]` $=$ 2.881, $b =$ `[prada2011ConcentrationB]` $=$ 1.257, $c =$ `[prada2011ConcentrationC]` $=$ 1.022, $d =$ `[prada2011ConcentrationD]` 0.060, $c_0 =$ `[prada2011ConcentrationC0]` $=$ 3.681, $c_1 =$ `[prada2011ConcentrationC1]` $=$ 5.033, $x_0 =$ `[prada2011Concentrat` 0.424, $x_1 =$ `[prada2011ConcentrationX1]` $=$ 0.526, $\sigma_0^{-1} =$ `[prada2011ConcentrationInverseSigma0]` $=$ 1.047, $\sigma_1^{-1} =$ `[prada2011ConcentrationInverseSigma1]` $=$ 1.646, $\alpha =$ `[prada2011ConcentrationAlpha]` $=$ 6.948, and $\beta =$ `[prada2011ConcentrationBeta]` $=$ 7.386.

Dutton & Macciò (2014)

Selected with `darkMatterProfileConcentrationMethod=duttonMaccio2014`, under this method the concentration is computed using a fitting function from Dutton and Macciò [2014]:

$$\log_{10} c = A + B \log_{10} M_{\rm halo}. \tag{12.89}$$

The parameters are a functon of redshift, z. We use the following fit suggested by Dutton and Macciò [2014] results:

$$A = A_1 + (A_2 - A_1) \exp[A_3 z^{A_4}]$$

$$B = B_1 + B_2 z. \tag{12.90}$$

The coefficients are chosen from one of the three sets given by Dutton and Macciò [2014], controlled via the `[duttonMaccio2014FitType]` parameter, as described in Table 12.3.

Fit type	Profile	$\Delta_{\rm vir}$	A_1	A_2	A_3	A_4	B_1	B_2
`nfwVirial`	NFW	Top-hat	+0.537	+1.025	−0.718	+1.080	−0.097	+0.024
`nfw200`	NFW	200	+0.520	+0.905	−0.617	+1.210	−0.101	+0.026
`einasto200`	Einasto	200	+0.459	+0.977	−0.490	+1.303	−0.130	+0.029

Table 12.3.: Coefficients appearing in the dark matter halo profile concentration fitting functions of Dutton and Macciò [2014]. The "fit type" is specified by the `[duttonMaccio2014FitType]` parameter.

Diemer & Kravtsov (2014)

Selected with `darkMatterProfileConcentrationMethod=diemerKravtsov2014`, under this method the concentration is computed using a fitting function from Diemer and Kravtsov [2014]:

$$c = \frac{c_{\rm min}}{2}\left[\left(\frac{\nu}{\nu_{\rm min}}\right)^{-\alpha} + \left(\frac{\nu}{\nu_{\rm min}}\right)^{\beta}\right], \tag{12.91}$$

where $c_{\rm min} = \phi_0 + \phi_1 n$, $\nu_{\rm min} = \eta_0 + \eta_1 n$, n is the logarithmic slope of the linear power spectrum at wavenumber $k = \kappa 2\pi/R$, R is the comoving Lagrangian radius of the halo, $R = [3M/4\pi\rho_{\rm M}(z=0)]^{1/3}$, and $\nu = \delta_{\rm crit}(t)/\sigma(M)$ is the peak height parameter. The numerical parameters $(\kappa, \phi_0, \phi_1, \eta_0, \eta_1, \alpha, \beta)$ are set by the parameters `[darkMatterProfileConcentrationDiemerKravtsov2014Kappa]`, `[darkMatterProfileConcentrationD`
`[darkMatterProfileConcentrationDiemerKravtsov2014Phi1]`, `[darkMatterProfileConcentrationDiemerKravtsov201`
`[darkMatterProfileConcentrationDiemerKravtsov2014Eta1]`, `[darkMatterProfileConcentrationDiemerKravtsov201`
`[darkMatterProfileConcentrationDiemerKravtsov2014Beta]`, respectively, and default to the values given in Table 3 of Diemer and Kravtsov [2014] for the median relation, namely $(0.69, 6.58, 1.37, 6.82, 1.42, 1.12, 1.69)$.

Schneider (2015)

Selected with `darkMatterProfileConcentrationMethod=schneider2015`, under this method the concentration using the algorithm of Schneider [2015]. Specifically, a reference model for concentrations in defined in a specific cosmological model. The concentration for a halo of given mass, M, and redshift, z_0, is then found by finding the redshift of collapse, $z_{\rm c}$ for the halo by solving:

$$\delta_{\rm c}(z_{\rm c}) = \left(\frac{\pi}{2}\left[\sigma^2(fM) - \sigma^2(M)\right]\right)^{1/2} + \delta_{\rm c})(z_0), \tag{12.92}$$

where $\delta_{\rm c}(z)$ is the critical overdensity for collapse at redshift z, and f is the fraction of a halo's mass assembled at formation time (given by the `[massFractionFormation]` parameter. From this, the mass of a halo in the reference model with the same redshift of collapse is found, and the reference model is used to compute the concentration of a halo of that mass.

WDM

Selected with `darkMatterProfileConcentrationMethod=WDM`, under this method the concentration is computed by applying the correction factor of Schneider et al. [2012]:

$$c_{\rm WDM} = c_{\rm CDM}\left[1 + \gamma_1\left(\frac{M_{1/2}}{M_{\rm halo}}\right)^{\gamma_2}\right]^{-1}, \tag{12.93}$$

where $\gamma_1 = 15$, $\gamma_2 = 0.3$, $M_{1/2}$ is the mass corresponding to the wavenumber at which the WDM transfer function is suppressed below the CDM transfer function by a factor of 2, and $M_{\rm halo}$ is the mass of the dark matter halo, to a CDM concentration algorithm as specified by `[darkMatterProfileConcentrationCDMMethod]`.

12.11.4. Density Profile Shape

The method used to compute any shape parameter of dark matter profiles is selected via the `darkMatterShapeMethod` parameter.

Gao (2008)

Selected with `darkMatterShapeMethod=Gao2008`, under this method the shape parameter for Einasto density profiles is computed using a fitting function from Gao et al. [2008]:

$$\alpha = \begin{cases} 0.155 + 0.0095\nu^2 & \text{if } \nu < 3.907 \\ 0.3 & \text{if } \nu \geq 3.907, \end{cases} \tag{12.94}$$

where $\nu = \delta_c(t)/\sigma(M)$ is the peak height of the halo. The truncation at $\alpha = 0.3$ is included since Gao et al. [2008]'s fits do not probe this region and extremely large values of α are numerically troublesome.

12.11.5. Mass Loss Rates

The method used to compute the rate of mass loss from dark matter (sub)halos is selected via the `darkMatterHaloMassLossRateMethod` parameter.

Null

Selected with `darkMatterHaloMassLossRateMethod=null`, this method assumes a zero rate of mass loss from dark matter halos.

van den Bosch et al. (2005)

Selected with `darkMatterHaloMassLossRateMethod=vanDenBosch2005`, this method uses the algorithm of van den Bosch et al. [2005] to compute the rate of mass loss. Specifically:

$$\dot{M}_{\rm node,bound} = -\frac{M_{\rm node,bound}}{\tau} \left(M_{\rm node,bound}/M_{\rm node,parent}\right)^{\zeta}, \tag{12.95}$$

where $M_{\rm node,parent}$ is the mass of the parent node in which the halo lives and

$$\tau = \tau_0 \left(\frac{\Delta_{\rm vir}(t)}{\Delta(t_0)}\right)^{-1/2} a^{3/2}, \tag{12.96}$$

where $\Delta_{\rm vir}(t)$ is the virial overdensity of halos at time t and a is the expansion factor. The fitting parameters, τ_0 and ζ have values of 0.13 Gyr and 0.36 respectively as determined by van den Bosch et al. [2005]. Note that van den Bosch et al. [2005] write this expression in a slightly different form since their $\Delta_{\rm vir}$ is defined relative to the critical density rather than the mean density as it is in Galacticus. In both cases, the timescale τ simply scales as $\langle\rho_{\rm vir}\rangle^{-1/2}$ where $\langle\rho_{\rm vir}\rangle$ is the mean virial overdensity of halos.

12.11.6. Mass Sampling Density Function

The method used to compute the halo mass density function is selected via the `haloMassFunctionSamplingMethod` parameter.

Power Law

Selected with `haloMassFunctionSamplingMethod=powerLaw`, under this method the sampling density function is given by

$$\gamma(M) = \log_{10}(M/M_{\rm minimum})^{-\alpha/(1+\alpha)}, \tag{12.97}$$

The resulting distribution of halo masses is such that the mass of the $i^{\rm th}$ halo is

$$M_{\rm halo,i} = \exp\left[\ln(M_{\rm halo,min}) + \ln\left(M_{\rm halo,max}/M_{\rm halo,min}\right) x_i^{1+\alpha}\right]. \tag{12.98}$$

Here, x_i is a number between 0 and 1 and α =`mergerTreeBuildTreesHaloMassExponent` is an input parameter that controls the relative number of low and high mass tree produced.

Halo Mass Function

Selected with `haloMassFunctionSamplingMethod=haloMassFunction`, this method sets the sampling density function equal to the halo mass function,

$$\gamma(M) = {\rm minmax}(\phi_{\rm min}, \phi_{\rm max}, {\rm d}n(M)/{\rm d}\log M[1 + p_1 \log_{10}(M/10^{13}M_\odot) + p_2 \log_{10}^2(M/10^{13}M_\odot)), \tag{12.99}$$

where $\phi_{\rm min}$ =`[haloMassFunctionSamplingAbundanceMinimum]`, $\phi_{\rm max}$ =`[haloMassFunctionSamplingAbundanceMaximum]`, p_1 =`[haloMassFunctionSamplingModifier1]`, p_2 =`[haloMassFunctionSamplingModifier2]`, and

$${\rm minmax}(a,b,x) = \begin{cases} a & \text{if } x < a \\ x & \text{if } a \le x \le b \\ b & \text{if } x > b, \end{cases} \tag{12.100}$$

resulting in a sample of halos representative of a volume of space.

Stellar Mass Function

Selected with `haloMassFunctionSamplingMethod=stellarMassFunction`, in this case the sampling density is chosen to give optimally minimal errors on the model stellar mass function (see §9.4.1 for full details). This calculation requires the observational errors on the stellar mass function to be known. A simple model of the form

$$\sigma^2(M) = \left[\phi_0 \left(\frac{M}{M_*}\right)^\alpha \exp\left(-\left[\frac{M}{M_*}\right]^\beta\right)\right]^2 + \sigma_0^2 \tag{12.101}$$

is used to model the error, $\sigma(M)$, on the observed stellar mass function as a function of stellar mass, M, where ϕ_0 =`[haloMassFunctionSamplingStellarMassFunctionErrorPhi0]`,M_* =`[haloMassFunctionSamplingStellarMas`
and σ_0 =`[haloMassFunctionSamplingStellarMassFunctionErrorConstant]`.

12.11.7. Spin Parameter Distribution

The method used to compute the distribution of dark matter halo spin parameters is selected via the `haloSpinDistributionMethod` parameter.

Lognormal

Selected with `haloSpinDistributionMethod=lognormal`, under this method the spin is drawn from a lognormal distribution with median `[lognormalSpinDistributionMedian]` and width `[lognormalSpinDistributionSigma]`

Bett et al. (2007)

Selected with `haloSpinDistributionMethod=Bett2007`, under this method the spin is drawn from the distribution found by Bett et al. [2007]. The λ_0 and α parameter of Bett et al.'s distribution are set by the `[spinDistributionBett2007Lambda0]` and `[spinDistributionBett2007Alpha]` input parameters.

Delta Function

Selected with `haloSpinDistributionMethod=deltaFunction`, under this method the spin is drawn from a delta function distribution, $P(\lambda) = \delta(\lambda - \lambda_0)$, where $\lambda_0 =$`[deltaFunctionSpinDistributionSpin]`, i.e. a fixed value of spin equal to λ_0 is returned.

12.12. Disk Stability/Bar Formation

The method uses to compute the bar instability timescale for galactic disks is selected via the `barInstabilityMethod` parameter.

12.12.1. Null

Selected with `barInstabilityMethod=null`, this method assumes no instability and so returns an infinite timescale, and no external driving torque.

12.12.2. Efstathiou, Lake & Negroponte

Selected with `barInstabilityMethod=ELN`, this method uses the stability criterion of Efstathiou et al. [1982] to estimate when disks are unstable to bar formation:

$$\epsilon \left(\equiv \frac{V_{\rm peak}}{\sqrt{{\rm G}M_{\rm disk}/r_{\rm disk}}} \right) < \epsilon_{\rm c}, \tag{12.102}$$

for stability, where $V_{\rm peak}$ is the peak velocity in the rotation curve (computed here assuming an isolated exponential disk), $M_{\rm disk}$ is the mass of the disk and $r_{\rm disk}$ is its scale length (assuming an exponential disk). The value of $\epsilon_{\rm c}$ is linearly interpolated in the disk gas fraction between values for purely gaseous and stellar disks as specified by `stabilityThresholdStellar` and `stabilityThresholdGaseous` respectively. For disks which are judged to be unstable, the timescale for bar formation is estimated to be

$$t_{\rm bar} = t_{\rm disk} \left(\frac{\epsilon_{\rm c} - \epsilon_{\rm iso}}{\epsilon_{\rm c} - \epsilon} \right)^2, \tag{12.103}$$

where $\epsilon_{\rm iso}$ is the value of ϵ for an isolated disk and $t_{\rm disk}$ is the disk dynamical time, defined as r/V, at one scale length. This form gives an infinite timescale at the stability threshold, reducing to a dynamical time for highly unstable disks, while also ensuring that the slope of $t_{\rm bar}$ is continuous at the instability threshold. This method returns zero external driving torque.

12.12.3. Efstathiou, Lake & Negroponte + Tidal Forces

Selected with `barInstabilityMethod=ELN+tidal`, this method uses the stability criterion of Efstathiou et al. [1982] to estimate when disks are unstable to bar formation, but includes an additional term due to external tidal forces:

$$\epsilon \left(\equiv \frac{V_{\rm peak}}{\sqrt{{\rm G}M_{\rm disk}/r_{\rm disk}} + \max(\mathcal{T}, 0)} \right) < \epsilon_{\rm c}, \tag{12.104}$$

for stability, where $V_{\rm peak}$ is the peak velocity in the rotation curve (computed here assuming an isolated exponential disk), $M_{\rm disk}$ is the mass of the disk, $r_{\rm disk}$ is its scale length (assuming an exponential disk), $\mathcal{T} = \mathcal{F} r_{\rm disk}^2$ is the external driving specific torque, and $\mathcal{F}$ is the external tidal field (evaluated using the selected method; see §12.42). The value of $\epsilon_{\rm c}$ is linearly interpolated in the disk gas fraction between values for purely gaseous and stellar disks as specified by `stabilityThresholdStellar` and `stabilityThresholdGaseous` respectively. For disks which are judged to be unstable, the timescale for bar formation is estimated to be

$$t_{\rm bar} = t_{\rm disk} \frac{\epsilon_{\rm c} - \epsilon_{\rm iso}}{\epsilon_{\rm c} - \epsilon}, \tag{12.105}$$

where $\epsilon_{\rm iso}$ is the value of ϵ for an isolated disk and $t_{\rm disk}$ is the disk dynamical time, defined as r/V, at one scale length. This form gives an infinite timescale at the stability threshold, reducing to a dynamical time for highly unstable disks.

12.13. Excursion Set Barrier

The functional form of the excursion set barrier is selected via the `excursionSetBarrierMethod` parameter.

12.13.1. Linear Barrier

Selected with `excursionSetBarrierMethod=linear` this method assumes a barrier:

$$B(S) = B_0 + B_1 S, \tag{12.106}$$

where $B_0 =$`[excursionSetBarrierConstantCoefficient]`, and $B_0 =$`[excursionSetBarrierLinearCoefficient]`.

12.13.2. Quadratic Barrier

Selected with `excursionSetBarrierMethod=quadratic` this method assumes a barrier:

$$B(S) = B_0 + B_1 S + B_2 S^2, \tag{12.107}$$

where $B_0 =$`[excursionSetBarrierConstantCoefficient]`, $B_0 =$`[excursionSetBarrierLinearCoefficient]`, and $B_2 =$`[excursionSetBarrierQuadraticCoefficient]`.

12.13.3. Critical Overdensity Barrier

Selected with `excursionSetBarrierMethod=criticalOverdensity` this method assumes a barrier equal to the critical linear theory overdensity for halo collpase (see §12.6.7).

12.14. Excursion Set Barrier First Crossing Distribution

The algorithm to be used when solving the excursion set barrier first crossing distribution problem is selected via the `excursionSetFirstCrossingMethod` parameter.

12.14.1. Linear Barrier

Selected with `excursionSetFirstCrossingMethod=linearBarrier` this method assumes the solution for a linear barrier. Specifically, the first crossing distribution is

$$f(S,t) = B(0,t)\exp(-B(S,t)^2/2S)/S/\sqrt{2piS}, \tag{12.108}$$

where $B(S,t)$ is the (assumed-to-be-linear-in-S) barrier at time t and variance S. The first crossing rate is computed using a finite difference approximation between two closely-spaced times. The non-crossing rate is zero.

12.14.2. Farahi

Selected with `excursionSetFirstCrossingMethod=Farahi` this method solves the barrier first crossing problem using the method of Farahi (see Benson et al. 2012), which proceeds by finding the solution to the integral equation:

$$1 = \int_0^S f(S')\mathrm{d}S' + \int_{-\infty}^{B(S)} P(\delta,S)\mathrm{d}\delta, \tag{12.109}$$

where $P(\delta,S)\mathrm{d}\delta$ is the probability for a trajectory to lie between δ and $\delta+\mathrm{d}\delta$ at variance S. In the absence of a barrier, $P(\delta,S)$ would be equal to $P_0(\delta,S)$ which is simply a Gaussian distribution with variance S:

$$P_0(\delta,S) = \frac{1}{\sqrt{2\pi S}}\exp\left(-\frac{\delta^2}{2S}\right). \tag{12.110}$$

Since the barrier absorbs any random walks which cross is at smaller S, the actual $P(\delta,S)$ must therefore be given by:

$$P(\delta,S) = P_0(\delta,S) - \int_0^S f(S')P_0[\delta - B(S'), S-S']\mathrm{d}S'. \tag{12.111}$$

In the second term on the right hand side of eqn. (12.111) represents the $P_0[\delta - B(S'), S-S']$ term represents the distribution of random trajectories orginating from the point $(S,B(S))$. The integral therefore gives the fraction of trajectories which crossed the barrier at $S < S'$ and which can now be found at (S,δ).

Using this result, we can rewrite eqn. (12.109):

$$1 = \int_0^S f(S')\mathrm{d}S' + \int_{-\infty}^{B(S)} \left[P_0(\delta,S) - \int_0^S f(S')P_0(\delta - B(S'), S-S')\mathrm{d}S')\right]\mathrm{d}\delta, \tag{12.112}$$

in general and, for the Gaussian distribution of eqn. (12.110):

$$1 = \int_0^S f(S')\mathrm{d}S' + \int_{-\infty}^{B(S)} \left[\frac{1}{\sqrt{2\pi S}}\exp\left(-\frac{\delta^2}{2S}\right) - \int_0^S f(S')\frac{1}{\sqrt{2\pi(S-S')}}\exp\left(-\frac{[\delta - B(S')]^2}{2(S-S')}\right)\mathrm{d}S'\right]\mathrm{d}\delta. \tag{12.113}$$

The integral over $\mathrm{d}\delta$ can be carried out analytically to give:

$$1 = \int_0^S f(S')\mathrm{d}S' + \mathrm{erf}\left[\frac{B(S)}{\sqrt{2S}}\right] - \int_0^S f(S')\mathrm{erf}\left[\frac{B(S)-B(S')}{\sqrt{2(S-S')}}\right]\mathrm{d}S''. \tag{12.114}$$

We now discretize eqn. (12.114). Specifically, we divide the S space into N intervals defined by the points:

$$S_i = \begin{cases} 0 & \text{if } i = 0 \\ \sum_0^{i-1}\Delta S_i & \text{if } i > 1. \end{cases} \tag{12.115}$$

Note that $f(0) = 0$ by definition, so $f(S_0) = 0$ always. We choose $\Delta S_i = S_{\max}/N$ (i.e. uniform spacing in S) when computing first crossing distributions, and $\Delta S_i \propto S_i$ (i.e. uniform spacing in $\log(S)$) when computing first crossing rates.

Discretizing the integrals in eqn. (12.114) gives:

$$\int_0^{S_j} f(S')\mathrm{d}S' = \sum_{i=0}^{j-1} \frac{f(S_i) + f(S_{i+1})}{2} \Delta S_i \tag{12.116}$$

and:

$$\int_0^{S_j} f(S')\mathrm{erf}\left[\frac{B(S) - B(S')}{\sqrt{2(S - S')}}\right] \mathrm{d}S' = \sum_{i=0}^{j-1} \frac{1}{2}\left(f(S_i)\mathrm{erf}\left[\frac{B(S_j) - B(S_i)}{\sqrt{2(S_j - S_i)}}\right] + f(S_{i+1})\mathrm{erf}\left[\frac{B(S_j) - B(S_{i+1})}{\sqrt{2(S_j - S_{i+1})}}\right]\right)\Delta S_i. \tag{12.117}$$

We can now rewrite eqn. (12.114) in discretized form:

$$1 = \sum_{i=0}^{j-1} \frac{f(S_i) + f(S_{i+1})}{2}\Delta S_i + \mathrm{erf}\left[\frac{B(S_j)}{\sqrt{2S_j}}\right] - \frac{1}{2}\sum_{i=0}^{j-1}\left(f(S_i)\mathrm{erf}\left[\frac{B(S_j) - B(S_i)}{\sqrt{2(S_j - S_i)}}\right] + f(S_{i+1})\mathrm{erf}\left[\frac{B(S_j) - B(S_{i+1})}{\sqrt{2(S_j - S_{i+1})}}\right]\right)\Delta S_i. \tag{12.118}$$

Solving eqn. (12.118) for $f(S_j)$:

$$\begin{aligned}\left(\frac{1}{2} - \frac{1}{2}\mathrm{erf}\left[\frac{B(S_j) - B(S_j)}{\sqrt{2(S_j - S_j)}}\right]\right)\Delta S_{j-1} f(S_j) \quad = \quad & 1 - \sum_{i=0}^{j-2} \frac{f(S_i) + f(S_{i+1})}{2}\Delta S_i - \frac{f(S_{j-1})}{2}\Delta S_{j-1} - \mathrm{erf}\left\{\frac{B(S_j)}{\sqrt{2S_j}}\right\} \\ & + \frac{1}{2}\sum_{i=0}^{j-2}\left(f(S_i)\mathrm{erf}\left\{\frac{[B(S_j) - B(S_i)]}{\sqrt{2(S_j - S_i)}}\right\} + f(S_{i+1})\mathrm{erf}\left\{\frac{[B(S_j) - B(S_{i+1})]}{\sqrt{2(S_j - S_{i+1})}}\right\}\right) \\ & + \frac{1}{2}f(S_{j-1})\mathrm{erf}\left\{\frac{[B(S_j) - B(S_{j-1})]}{\sqrt{2(S_j - S_{j-1})}}\right\}\Delta S_{j-1}. \end{aligned} \tag{12}$$

For all barriers that we consider:

$$\mathrm{erf}\left[\frac{B(S_j) - B(S_j)}{\sqrt{2(S_j - S_j)}}\right] = 0. \tag{12.120}$$

We can then simplify eqn. (12.119):

$$\begin{aligned} f(S_j) \quad = \quad & \frac{2}{\Delta S_{j-1}}\left[1 - \sum_{i=0}^{j-2} \frac{f(S_i) + f(S_{i+1})}{2}\Delta S_i - \frac{f(S_{j-1})}{2}\Delta S_{j-1} - \mathrm{erf}\left\{\frac{B(S_j)}{\sqrt{2S_j}}\right\}\right. \\ & + \frac{1}{2}\sum_{i=0}^{j-2}\left(f(S_i)\mathrm{erf}\left\{\frac{[B(S_j) - B(S_i)]}{\sqrt{2(S_j - S_i)}}\right\} + f(S_{i+1})\mathrm{erf}\left\{\frac{[B(S_j) - B(S_{i+1})]}{\sqrt{2(S_j - S_{i+1})}}\right\}\right)\Delta S_i \\ & \left. + \frac{1}{2}f(S_{j-1})\mathrm{erf}\left\{\frac{[B(S_j) - B(S_{j-1})]}{\sqrt{2(S_j - S_{j-1})}}\right\}\Delta S_{j-1}\right]. \end{aligned} \tag{12.121}$$

Consolidating terms in the summations:

$$f(S_j) = \frac{2}{\Delta S_{j-1}}\left[1 - \mathrm{erf}\left\{\frac{B(S_j)}{\sqrt{2S_j}}\right\} - \sum_{i=0}^{j-1}\left\{1 - \mathrm{erf}\left[\frac{B(S_j) - B(S_i)}{\sqrt{2(S_j - S_i)}}\right]\right\} f(S_i)\frac{\Delta S_{i-1} + \Delta S_i}{2}\right]. \tag{12.122}$$

In the case of constant $\Delta S_i (= \Delta S)$ this can be simplified further:

$$f(S_j) = \frac{2}{\Delta S}\left[1 - \mathrm{erf}\left\{\frac{B(S_j)}{\sqrt{2S_j}}\right\}\right] - 2\sum_{i=0}^{j-1}\left\{1 - \mathrm{erf}\left[\frac{B(S_j) - B(S_i)}{\sqrt{2(S_j - S_i)}}\right]\right\} f(S_i). \tag{12.123}$$

In either case (i.e. eqns. 12.122 and 12.123) solution proceeds recursively: $f(S_0) = 0$ by definition, $f(S_1)$ depends only on the known barrier and $f(S_0)$, $f(S_j)$ depends only on the known barrier and $f(S_{<j})$.

The first crossing rate is computed using the same method but with an effective barrier which is offset by the position of the progenitor in the (δ, S) plane, plus a small shift in time. The non-crossing rate is computed directly by integrating over the first crossing rate distribution. Note that since the numerical integration occurs only up to a finite maximum S, a non-zero non-crossing rate will be computed for CDM-like barriers even though in reality they should have zero non-crossing rate. As such, use of this method for such barriers is not recommended.

12.14.3. Zhang & Hui (2006)

Selected with `excursionSetFirstCrossingMethod=ZhangHui2006` this method solves the barrier first crossing problem using the method of Zhang and Hui [2006]. First crossing (and non-crossing) rates are not supported by this method.

12.14.4. Zhang & Hui (2006) Higher Order

Selected with `excursionSetFirstCrossingMethod=ZhangHui2006HighOrder` this method solves the barrier first crossing problem using a higher-order extension of the method of Zhang and Hui [2006]. First crossing (and non-crossing) rates are not supported by this method.

In this method we discretize the first-crossing distribution function, and use a closed Newton-Cotes method to perform integrations. First, we mesh the S space using uniform spacing in S:

$$S_i = i \times \Delta S, i = 0, 1, \ldots, N, \Delta S = \frac{S}{N}. \tag{12.124}$$

Then we discretize the integral equation by Boole's rule. The integral equation becomes a set of linear algebraic equations:

$$\begin{aligned} f(S_i) &= g_1(S_i) + \frac{\Delta S}{90}\sum_{j=0:4}^{i-4}\{7f(S_j)g_2(S_i,S_j) + 32f(S_{j+1})g_2(S_i,S_{j+1}) \\ &\quad + 12f(S_{j+2})g_2(S_i,S_{j+2}) + 32f(S_{j+3})g_2(S_i,S_{j+3}) + 7f(S_{j+4})g_2(S_i,S_{j+4})\} \end{aligned} \tag{12.125}$$

Since $g_2(S, S')$ approaches infinity when S approaches S', one needs to define $g_2(S_i, S_i)$ carefully when $j = i$. We can rewrite the equation:

$$\begin{aligned} f(S_i) &= g_1(S_i) + \frac{4\Delta S}{90}\sum_{j=0:4}^{i-8}\{7f(S_j)g_2(S_i,S_j) + 32f(S_{j+1})g_2(S_i,S_{j+1}) \\ &\quad + 12f(S_{j+2})g_2(S_i,S_{j+2}) + 32f(S_{j+3})g_2(S_i,S_{j+3}) + 7f(S_{j+4})g_2(S_i,S_{j+4})\} \\ &\quad + \frac{4\bar{g}_2(S_i)\Delta S}{90}\Big(7f(S_{i-4}) + 32f(S_{i-3}) + 12f(S_{i-2}) + 32f(S_{i-1}) + 7f(S_i)\Big). \end{aligned} \tag{12.126}$$

For $\bar{g}_2(S_i)$ we have:

$$\bar{g}_2(S_i) = \frac{1}{\delta S}\int_{S-\delta S}^{S} g_2(S, S')\mathrm{d}S'. \tag{12.127}$$

In the above, δS depends on the range of the previous integral part. Generally, δS is equal to $4\Delta S$. The above equation can be solved for $f(S_i)$, giving:

$$\begin{aligned} f(S_i) &= \Bigg(g_1(S_i) + \frac{4\Delta S}{90} \sum_{j=0:4}^{i-8} \{7f(S_j)g_2(S_i,S_j) + 32f(S_{j+1})g_2(S_i,S_{j+1}) \\ &\quad + 12f(S_{j+2})g_2(S_i,S_{j+2}) + 32f(S_{j+3})g_2(S_i,S_{j+3}) + 7f(S_{j+4})g_2(S_i,S_{j+4})\} \\ &\quad + \frac{4\bar{g}_2(S_i)\Delta S}{90}\Big(7f(S_{i-4}) + 32f(S_{i-3}) + 12f(S_{i-2}) + 32f(S_{i-1})\Big)\Bigg) \\ &\quad \times \left(1 - \frac{4\bar{g}_2(S_i)\Delta S}{90}\right)^{-1}. \end{aligned} \tag{12.128}$$

Not all of the i's are divisible by 4. So, for the first m^{th} spaces, we need to calculate the integral part, separately, where m is the remainder of i by the modulus of 4. It is a good approximation to calculate the first part linearly. Consequently, the final formula for the general problem is:

$$\begin{aligned} f(S_i) &= \Bigg(g_1(S_i) + \frac{\Delta S}{2} \sum_{j=0}^{m-1} \Big(f(S_j)g_2(S_i,S_j) + f(S_{j+1})g_2(S_i,S_{j+1})\Big) \\ &\quad + \frac{4\Delta S}{90} \sum_{\frac{j-m}{4}=0}^{i-8} \{7f(S_j)g_2(S_i,S_j) + 32f(S_{j+1})g_2(S_i,S_{j+1}) \\ &\quad + 12f(S_{j+2})g_2(S_i,S_{j+2}) + 32f(S_{j+3})g_2(S_i,S_{j+3}) + 7f(S_{j+4})g_2(S_i,S_{j+4})\} \\ &\quad + \frac{4\bar{g}_2(S_i)\Delta S}{90}\Big(7f(S_{i-4}) + 32f(S_{i-3}) + 12f(S_{i-2}) + 32f(S_{i-1})\Big)\Bigg) \\ &\quad \times \left(1 - \frac{4\bar{g}_2(S_i)\Delta S}{90}\right)^{-1}. \end{aligned} \tag{12.129}$$

Finally, we can solve the first-crossing distribution function, giving:

$$f(S_0) = g_1(S_0) = 0 \tag{12.130}$$

$$f(S_1) = g_1(S_1)\left(1 - \frac{\bar{g}_2(S_1)\Delta S}{2}\right)^{-1} \tag{12.131}$$

$$f(S_2) = \left(g_1(S_2) + \frac{\Delta S}{2} 2\bar{g}_2(S_2) f(S_1)\right)\left(1 - \frac{\bar{g}_2(S_2)\Delta S}{2}\right)^{-1} \tag{12.132}$$

$$f(S_3) = \left(g_1(S_2) + \frac{\Delta S}{2} 2\bar{g}_2(S_3)(f(S_1) + f(S_2))\right)\left(1 - \frac{\bar{g}_2(S_3)\Delta S}{2}\right)^{-1} \tag{12.133}$$

$$\begin{aligned} f(S_i) = & \left(g_1(S_i) + \frac{\Delta S}{2}\sum_{j=0}^{m-1}\left(f(S_j)g_2(S_i,S_j) + f(S_{j+1})g_2(S_i,S_{j+1})\right)\right. \\ & + \frac{4\Delta S}{90}\sum_{\frac{j-m}{4}=0}^{i-8}\{7f(S_j)g_2(S_i,S_j) + 32f(S_{j+1})g_2(S_i,S_{j+1}) \\ & + 12f(S_{j+2})g_2(S_i,S_{j+2}) + 32f(S_{j+3})g_2(S_i,S_{j+3}) + 7f(S_{j+4})g_2(S_i,S_{j+4})\} \\ & \left. + \frac{4\bar{g}_2(S_i)\Delta S}{90}\left(7f(S_{i-4}) + 32f(S_{i-3}) + 12f(S_{i-2}) + 32f(S_{i-1})\right)\right) \\ & \times\left(1 - \frac{4\bar{g}_2(S_i)\Delta S}{90}\right)^{-1}, \end{aligned} \tag{12.134}$$

where $\bar{g}_2(S_1)$ is defined as:

$$\bar{g}_2(S_1) = \frac{1}{\Delta S}\int_0^{S_1} g_2(S,S')\mathrm{d}S' \text{ if } i = 1 \tag{12.135}$$

$$\bar{g}_2(S_2) = \frac{1}{2\Delta S}\int_0^{S_2} g_2(S,S')\mathrm{d}S' \text{ if } i = 0 \tag{12.136}$$

$$\bar{g}_2(S_3) = \frac{1}{3\Delta S}\int_0^{S_3} g_2(S,S')\mathrm{d}S' \text{ if } i = 0 \tag{12.137}$$

$$\bar{g}_2(S_i) = \frac{1}{4\Delta S}\int_{S_i-4\Delta S}^{S_i} g_2(S,S')\mathrm{d}S' \text{ if } i > 3. \tag{12.138}$$

The error term for this method of discretization is:

$$\epsilon = \frac{(\delta S)^7}{1935360} f^{(6)}(\xi), \tag{12.139}$$

where $f^{(6)}(\xi)$ is the absolute maximum of the sixth derivative in the range of $[S_i, S_i + \delta S]$. For this problem, $\delta S = 4\Delta S$, so:

$$\epsilon \leq \frac{(\Delta S)^7}{118.125} f^{(6)}(\xi). \tag{12.140}$$

Since there are $N/4$ intervals, the maximum deviation from the real value of the function is:

$$\epsilon \leq \sum_{i=1}\frac{N}{4}\frac{(\Delta S)^7}{118.125} f^{(6)}(\xi_i) \leq N4\frac{(\Delta S)^7}{472.5} f^{(6)}(\xi),$$

where $f^{(6)}(\xi)$ is the absolute maximum of the sixth derivative in the domain, $[0, S]$.

12.15. Excursion Set Barrier Remapping

Remappins of the excursion set barrier are selected via the `excursionSetBarrierRemapMethods` (for calculations of first crossing distributions) and `excursionSetBarrierRatesRemapMethods` (for calculations of first crossing rate distributions) parameters. The parameters may specify multiple remappings—these will be applied to the barrier in order.

12.15.1. Null Remapping

Selected with `excursionSetBarrier(Rates)RemapMethods=null` this method leaves the barrier unmodified.

12.15.2. Scale Remapping

Selected with `excursionSetBarrier(Rates)RemapMethods=scale` this method multiplies the barrier by `[excursionSetBarrierRemapScalingFactor]`.

12.15.3. Sheth-Mo-Tormen Remapping

Selected with `excursionSetBarrier(Rates)RemapMethods=Sheth-Mo-Tormen` this method remaps the barrier according to the algorithm of Sheth et al. [2001]:

$$B(S) \rightarrow \sqrt{A}B(S)\left(1+b\left[\frac{S}{AB^2(S)}\right]^c\right), \tag{12.141}$$

where $A=0.707$, $b=0.5$, and $c=0.6$.

12.16. Galactic Structure

The algorithm to be used when solving for galactic structure (specifically, finding radii of galactic components) is selected via the `galacticStructureRadiusSolverMethod` parameter.

12.16.1. Fixed

Selected with `galacticStructureRadiusSolverMethod=fixed` this method determines the sizes of galactic components by assuming that radius equals

$$r = f_\mathrm{r}\lambda r_0 \tag{12.142}$$

where r_0 is the virial or turnaround radius of the node if `[galacticStructureRadiiFixedRadius]=virialRadius` or `turnaround` respectively, λ is its spin parameter and $f_\mathrm{r}=$`[galacticStructureRadiiFixedFactor]` is a parameter.

12.16.2. Linear

Selected with `galacticStructureRadiusSolverMethod=linear` this method determines the sizes of galactic components by assuming that radius scales linearly with specific angular momentum such that

$$r = r_\mathrm{vir} j / j_\mathrm{vir} \tag{12.143}$$

where j is the specific angular momentum of the component (at whatever point in the profile is to be solved for), r is radius, r_vir is the virial radius of the node and $j_\mathrm{vir} = r_\mathrm{vir} v_\mathrm{vir}$ with v_vir being the virial velocity of the node.

12.16.3. Simple

Selected with `galacticStructureRadiusSolverMethod=simple` this method determines the sizes of galactic components by assuming that their self-gravity is negligible (i.e. that the gravitational potential well is dominated by dark matter) and that, therefore, baryons do not modify the dark matter density profile. The radius of a given component is then found by solving

$$j = \sqrt{\mathrm{G} M_\mathrm{DM}(r) r}, \tag{12.144}$$

where j is the specific angular momentum of the component (at whatever point in the profile is to be solved for), r is radius and $M(r)$ is the mass of dark matter within radius r. The parameter `[adiabaticContractionUseFormationHalo]` controls whether the structure of the galaxy will be solved for using the properties of its present node or those of its node at the time of node formation (which requires that "node formation" has been suitably defined and implemented by a component).

12.16.4. Adiabatic

Selected with `galacticStructureRadiusSolverMethod=adiabatic`, this method takes into account the baryonic self-gravity of all galactic components when solving for structure and additionally accounts for backreaction of the baryons on the dark matter density profile. Solution proceeds via an iterative procedure to find equilibrium radii for all galaxies in a consistently contracted halo. To account for adiabatic contraction the mass of dark matter within a given radius r_f is taken to be equal to that originally within radius r_i in the uncontracted halo. This initial radius is computed using the selected galactic structure solver initial radius algorithm (see §12.17). The parameter `[adiabaticContractionUseFormationHalo]` controls whether the structure of the galaxy will be solved for using the properties of its present node or those of its node at the time of node formation (which requires that "node formation" has been suitably defined and implemented by a component).

12.17. Galactic Structure Initial Radii

The algorithm to be used when solving for the initial radius in the dark matter halo when computing galactic structure is selected via the `galacticStructureRadiusSolverInitialRadiusMethod` parameter.

12.17.1. Static

Selected with `galacticStructureRadiusSolverInitialRadiusMethod=static` this method assumes that the dark matter halo is static and so sets the initial radius equal to the final radius.

12.17.2. Adiabatic

Selected with `galacticStructureRadiusSolverInitialRadiusMethod=adiabatic` this method assumes that adiabatic contraction follows the algorithm of Gnedin et al. [2004]. The parameters A and ω of that model are specified via input parameters `adiabaticContractionGnedinA` and `adiabaticContractionGnedinOmega` respectively.

Given the final radius, r_f, the corresponding initial radius, r_i, is found by solving:

$$f_\mathrm{i} M_\mathrm{total,0}(\bar{r}_\mathrm{i}) r_\mathrm{i} = f_\mathrm{f} M_\mathrm{total,0}(\bar{r}_\mathrm{i}) r_\mathrm{f} + V_\mathrm{b}^2(\bar{r}_\mathrm{f}) \bar{r}_\mathrm{f} r_\mathrm{f} / \mathrm{G}, \tag{12.145}$$

where $M_\mathrm{total,0}(r)$ is the initial total matter profile, $V_\mathrm{b}(r)$ is the baryonic contribution to the rotation curve, f_i, is the fraction of mass within the virial radius compared to the node mass[1], $f_\mathrm{f} = (\Omega_\mathrm{M} -$

[1] In Galacticus the "node mass" refers to the total mass of the node, assuming it has the universal complement of baryons. Since some halos may contain less than the complete complement of baryons it is possible that $f_\mathrm{i} < 1$.

$\Omega_\mathrm{b})/\Omega_\mathrm{M} + M_\mathrm{satellite,baryonic}/M_\mathrm{total}$, $M_\mathrm{satellite,baryonic}$ is the baryonic mass in any satellite halos, M_total is the node mass, and

$$\frac{\bar{r}}{r_\mathrm{vir}} = A\left(\frac{r}{r_\mathrm{vir}}\right)^{\omega}, \tag{12.146}$$

where r_vir is the virial radius. Note that we explicitly assume that the initial, uncontracted total density profile has the same shape as the initial dark matter density profile, that contraction of the halo occurs with no shell crossing, and that satellite halos trace the dark matter profile of their host halo.

The derivative, $\mathrm{d}r_\mathrm{f}/\mathrm{d}d_\mathrm{i} \equiv r'_\mathrm{i}$ is found by taking the derivative of eqn. (12.145) to give:

$$\begin{aligned} & f_\mathrm{i} M_\mathrm{total,0}(\bar{r}_\mathrm{i}) r'_\mathrm{i} + f_\mathrm{i} 4\pi \bar{r}_\mathrm{i}^2 \rho_\mathrm{total,0}(\bar{r}_\mathrm{i}) \frac{\mathrm{d}\bar{r}_\mathrm{i}}{\mathrm{d}r_\mathrm{i}} r_\mathrm{i} r'_\mathrm{i} \\ = \;& f_\mathrm{f} M_\mathrm{total,0}(\bar{r}_\mathrm{i}) + f_\mathrm{i} 4\pi \bar{r}_\mathrm{i}^2 \rho_\mathrm{total,0}(\bar{r}_\mathrm{i}) \frac{\mathrm{d}\bar{r}_\mathrm{i}}{\mathrm{d}r_\mathrm{i}} r_\mathrm{f} r'_\mathrm{i} \\ + \;& V_\mathrm{b}^2(\bar{r}_\mathrm{f})\bar{r}_\mathrm{f}/\mathrm{G} + V_\mathrm{b}^2(\bar{r}_\mathrm{f}) \frac{\mathrm{d}\bar{r}_\mathrm{f}}{\mathrm{d}r_\mathrm{f}} r_\mathrm{f}/\mathrm{G} + \frac{\mathrm{d}V_\mathrm{b}^2}{\mathrm{d}\bar{r}_\mathrm{f}}(\bar{r}_\mathrm{f}) \frac{\mathrm{d}\bar{r}_\mathrm{f}}{\mathrm{d}r_\mathrm{f}} \bar{r}_\mathrm{f} r_\mathrm{f}/\mathrm{G}, \end{aligned} \tag{12.147}$$

where

$$\frac{\mathrm{d}\bar{r}}{\mathrm{d}r} = A\left(\frac{r}{r_\mathrm{vir}}\right)^{\omega-1}, \tag{12.148}$$

and which can then be solved numerically for r'_i.

12.18. Galaxy Merging

The process of merging two galaxies currently involves two algorithms: one which decides how the merger causes mass components from both galaxies to move and one which determines the size of the remnant galaxy spheroid.

12.18.1. Mass Movements

The movement of mass elements in the merging galaxies is determined by the `satelliteMergingMassMovementsMethod` parameter.

Very Simple

Selected with `satelliteMergingMassMovementsMethod=verySimple`, this method assumes that the satellite material is always added to the disk of the host, while the host mass is not moved.

Simple

Selected with `satelliteMergingMassMovementsMethod=simple`, this method implements mass movements according to:

- If $M_\mathrm{satellite} > f_\mathrm{major} M_\mathrm{central}$ then all mass from both satellite and central galaxies moves to the spheroid component of the central galaxy;
- Otherwise: Gas from the satellite moves to the component of the central specified by the `[minorMergerGasMovesTo]` parameter (either "`disk`" or "`spheroid`"), stars from the satellite moves to the spheroid of the central and mass in the central does not move.

Here, f_major = `[majorMergerMassRatio]` is the mass ratio above which a merger is considered to be "major".

Baugh et al. (2005)

Selected with `satelliteMergingMassMovementsMethod=Baugh2005`, this method implements mass movements according to:

- If $M_{\rm satellite} > f_{\rm major} M_{\rm central}$ then all mass from both satellite and central galaxies moves to the spheroid component of the central galaxy;
- Otherwise:
  - If $M_{\rm central,spheroid} < f_{\rm burst} M_{\rm central}$ and the gas fraction in the host equals or exceeds $f_{\rm gas,crit}$ then all gas is moved to the host spheroid, while the host stellar disk remains in place.
  - Otherwise, gas from the satellite moves to the component of the central specified by the `[minorMergerGasMovesTo]` parameter (either "`disk`" or "`spheroid`"), stars from the satellite moves to the spheroid of the central and mass in the central does not move.

Here, $f_{\rm major}$ =`[majorMergerMassRatio]` is the mass ratio above which a merger is considered to be "major", while $f_{\rm burst}$ =`[burstMassRatio]` and $f_{\rm gas,crit}$ =`[burstCriticalGasFraction]`.

12.18.2. Remnant Sizes

The method used to calculate the sizes of merger remnant spheroids is selected by the `satelliteMergingRemnantSizeMethod` parameter.

Null

Selected using `satelliteMergingRemnantSizeMethod=null`, this is a null method which does nothing at all. It is useful, for example, when running Galacticus to study dark matter only (i.e. when no galaxy properties are computed).

Cole et al. (2000)

Selected using `satelliteMergingRemnantSizeMethod=Cole2000`, this method uses the algorithm of Cole et al. [2000] to compute merger remnant spheroid sizes. Specifically

$$\frac{(M_1+M_2)^2}{r_{\rm new}} = \frac{M_1^2}{r_1} + \frac{M_2^2}{r_2} + \frac{f_{\rm orbit}}{c}\frac{M_1 M_2}{r_1+r_2}, \tag{12.149}$$

where M_1 and M_2 are the baryonic masses of the components of the merging galaxies that will end up in the spheroid component of the remnant[2] and r_1 and r_2 are the half mass radii of those same components of the merging galaxies[3], $r_{\rm new}$ is the half mass radius of the spheroidal component of the remnant galaxy and c is a constant which depends on the distribution of the mass. For a Hernquist spheroid $c = 0.40$ can be found by numerical integration while for a exponential disk $c = 0.49$. For simplicity a value of $c = 0.5$ is adopted for all components. The parameter $f_{\rm orbit}$ =`mergerRemnantSizeOrbitalEnergy` depends on the orbital parameters of the galaxy pair. For example, a value of $f_{\rm orbit} = 1$ corresponds to point mass galaxies in circular orbits about their center of mass.

A subtelty arises because the above expression accounts for only the baryonic mass of material which becomes part of the spheroid component of the remnant. In reality, there are additional terms in the energy equation due to the interaction of this material with any dark matter mass in each galaxy and any

[2] Depending on the merging rules (see §12.18.1) not all mass may be placed into the spheroid component of the remnant.
[3] In practice, Galacticus computes a weighted average of the disk and spheroid half-mass radii of each galaxy, with weights equal to the masses of each component (disk and spheroid) which will become part of the spheroid component of the remnant.

baryonic mass of each galaxy which does not become part of the spheroid component of the remnant. To account for this additional matter, an effective boost factor, $f_{\rm boost}$, to the specific angular momentum of each component of each merging galaxy is computed:

$$f_{\rm boost} = \frac{j}{\sqrt{\mathrm{G}Mr_{1/2}}}, \tag{12.150}$$

where j is the specific angular momentum of the component, M is its total baryonic mass and $r_{1/2}$ is its half-mass radius. The mass-weighted mean boost factor is found by combining those of all components which will form part of the spheroid of the remnant. The final specific angular momentum of the remnant spheroid is then given by:

$$j_{\rm new} = \langle f_{\rm boost} \rangle r_{\rm new} V_{\rm new}, \tag{12.151}$$

where

$$V_{\rm new}^2 = \frac{\mathrm{G}(M_1+M_2)}{r_{\rm new}}. \tag{12.152}$$

Covington et al. (2008)

Selected using `satelliteMergingRemnantSizeMethod=Covington2008`, this method uses the algorithm of Covington et al. [2008] to compute merger remnant spheroid sizes. Specifically

$$\frac{(M_1+M_2)^2}{r_{\rm new}} = \left[\frac{M_1^2}{r_1} + \frac{M_2^2}{r_2} + \frac{f_{\rm orbit}}{c}\frac{M_1 M_2}{r_1+r_2}\right](1+f_{\rm gas}C_{\rm rad}), \tag{12.153}$$

where M_1 and M_2 are the baryonic masses of the merging galaxies and r_1 and r_2 are their half mass radii, $r_{\rm new}$ is the half mass radius of the spheroidal component of the remnant galaxy and c is a constant which depends on the distribution of the mass. For a Hernquist spheroid $c = 0.40$ can be found by numerical integration while for a exponential disk $c = 0.49$. For simplicity a value of $c = 0.5$ is adopted for all components. The parameter $f_{\rm orbit}$ =`mergerRemnantSizeOrbitalEnergy` depends on the orbital parameters of the galaxy pair. For example, a value of $f_{\rm orbit} = 1$ corresponds to point mass galaxies in circular orbits about their center of mass. The final term on the right hand side of eqn. (12.153) gives a correction to the final energy of the remnant due to dissipational losses based on the results of Covington et al. [2011], with

$$f_{\rm gas} = \frac{M_{1,\rm gas}+M_{2,\rm gas}}{M_1+M_2} \tag{12.154}$$

begin the gas fraction of the progenitor galaxies. By default, $C_{\rm rad} = 2.75$ [Covington et al., 2011]. To account for the effects of dark matter and non-spheroid baryonic matter the same approach is used as in the Cole et al. [2000] algorithm (see §12.18.2).

12.18.3. Progenitor Properties

The method used to calculate the properties of merger progenitor galaxies is selected by the `satelliteMergingRemnantProgen` parameter.

Cole et al. (2000)

Selected using `satelliteMergingRemnantProgenitorPropertiesMethod=Cole2000`, this method uses the algorithms of Cole et al. [2000] to compute progenitor properties. Masses of progenitors are set to

$$M_{\rm host|satellite} = \sum_{i=\rm disk|spheroid} \sum_{j=\rm stars|gas} M_{i,j}, \tag{12.155}$$

where $M_{i,j}$ is the mass of mass type j in component i. Masses of progenitors that will end up in the remnant spheroid are set to

$$M_{\rm spheroid\ host|satellite} = \sum_{i={\rm disk|spheroid}} \sum_{j={\rm stars|gas}} M_{i,j}\delta_{i,j}, \tag{12.156}$$

where $\delta_{i,j} = 0$ of mass type j in component i will end up in the remnant spheroid and 0 otherwise. Radii of material that will end up in the spheroid are set by finding the solution to:

$$\sum_{i={\rm disk|spheroid}} \sum_{j={\rm stars|gas}} M_{i,j}(r)\delta_{i,j} = \frac{1}{2} \sum_{i={\rm disk|spheroid}} \sum_{j={\rm stars|gas}} M_{i,j}\delta_{i,j}, \tag{12.157}$$

such that the radii are the half-mass radii of the material that will end up in the remnant spheroid. Finally, the angular momentum factor is set to

$$f_{\rm AM\ host|satellite} = \frac{1}{M_{\rm spheroid\ host|satellite}} \sum_{i={\rm disk|spheroid}} \sum_{j={\rm stars|gas}} M_{i,j} \frac{J_{i,j}}{{\rm G}M_{i,j}^{3/2} r_{1/2\ i,j}} \delta_{i,j}, \tag{12.158}$$

where $J_{i,j}$ is the angular momentum or pseudo-angular momentum of mass type j in component i[4].

Standard

Selected using `satelliteMergingRemnantProgenitorPropertiesMethod=standard`, this is the standard method to compute progenitor properties. Masses of progenitors are set to

$$M_{\rm host|satellite} = \sum_{i={\rm disk|spheroid}} \sum_{j={\rm stars|gas}} M_{i,j}, \tag{12.159}$$

where $M_{i,j}$ is the mass of mass type j in component i. Masses of progenitors that will end up in the remnant spheroid are set to

$$M_{\rm spheroid\ host|satellite} = \sum_{i={\rm disk|spheroid}} \sum_{j={\rm stars|gas}} M_{i,j}\delta_{i,j}, \tag{12.160}$$

where $\delta_{i,j} = 0$ of mass type j in component i will end up in the remnant spheroid and 0 otherwise. Radii of material that will end up in the spheroid are set to

$$r_{\rm host|satellite} = \frac{1}{M_{\rm spheroid\ host|satellite}} \sum_{i={\rm disk|spheroid}} \sum_{j={\rm stars|gas}} M_{i,j} r_{1/2\ i,j} \delta_{i,j}. \tag{12.161}$$

Finally, the angular momentum factor is set to

$$f_{\rm AM\ host|satellite} = \frac{1}{M_{\rm spheroid\ host|satellite}} \sum_{i={\rm disk|spheroid}} \sum_{j={\rm stars|gas}} M_{i,j} \frac{J_{i,j}}{{\rm G}M_{i,j}^{3/2} r_{1/2\ i,j}} \delta_{i,j}, \tag{12.162}$$

where $J_{i,j}$ is the angular momentum or pseudo-angular momentum of mass type j in component i.

[4] This is technically not quite what Cole et al. [2000] do. Instead, when computing the masses of the material which ends up in the spheroid they include twice the mass of dark matter (accounting for the effects of adiabatic contraction) within the half-mass radius of each galaxy (as calculated above). The final angular momentum is then $j = \sqrt{{\rm G}M_{\rm remnant} r_{\rm remnant}/2}$ (where $M_{\rm remnant}$ includes the contribution from dark matter and the factor of 2 appears to make this the half-mass). This approach is currently not used in Galacticus since there is no way to get the mass of dark matter enclosed accounting for adiabatic contraction in the general case. This is a solvable problem, and so this algorithm is expected to be modified to match that of Cole et al. [2000] precisely in a future version of Galacticus.

12.19. Gravitational Lensing

The method to be used for computing the effects of gravitational lensing by large-scale structure is specified by the `gravitationalLensingMethod` parameter.

12.19.1. Takahashi et al. (2011)

Selected with `gravitationalLensingMethod=takahashi2011`, this method utilizes the fitting functions of Takahashi et al. [2011] to compute the effects of gravitational lensing. Specifically, eqn. 11 of Takahashi et al. [2011] is used. The parameters $\kappa_{\rm empty}$ and $\langle\kappa^2\rangle$ are computed from the assumed cosmology and non-linear power spectrum as described by [Takahashi et al., 2011, eqns. 5 and 2 respectively]. The parameters, N_κ, A_κ, and ω_κ of the lensing convergence distribution are determined using the conditions given by [Takahashi et al., 2011, eqn. 9].

12.19.2. Baryonic Modifier

Selected with `gravitationalLensingMethod=baryonicModifier`, this method modifies another gravitational lensing class to approximately account for the effects of baryons on the magnification distribution. The distribution to modify is specified via the `[gravitationalLensingBaryonicModifierOriginalDistribution]` parameter. The modification takes the form:

$$P(\mu) \rightarrow P(\mu) + \min[\alpha, \beta P(\mu)] \tag{12.163}$$

where $\alpha =$`[gravitationalLensingBaryonicModifierAlpha]` and $\beta =$`[gravitationalLensingBaryonicModifierBeta]`. The distribution is then renormalized to ensure that the cumulative probability reaches unity for infinite magnification. As an example, values of $\alpha = 2.05 \times 10^{-3}$ and $\beta = 0.62$ approximately reproduce the results of [Hilbert et al., 2008, their Fig. 1].

12.20. Halo Model Power Spectra Modifiers

The method to be used for modifying power spectra in halo model clustering calculations is specified by the `haloModelPowerSpectrumModifierMethod` parameter.

12.20.1. Null

Selected with `haloModelPowerSpectrumModifierMethod=null`, this method provides no modification.

12.20.2. Traxiality

Selected with `haloModelPowerSpectrumModifierMethod=triaxiality`, this method provides attempts to modify power spectra to approximately account for the effects of halo triaxiality using the results of Smith and Watts [2005]. Specifically, the one- and two-halo power spectra are multiplied by a correction factor, $\Delta^2_{\rm triax}/\Delta^2_{\rm sphere}$, derived from the lower panels of Figures 3 and 2 of Smith and Watts [2005] respectively for their "JS02" profile model. Given the uncertainty in this correction, the power spectrum covariance (if provided) is incremented by $\epsilon^2([\Delta^2_{\rm triax}/\Delta^2_{\rm sphere} - 1] \otimes [\Delta^2_{\rm triax}/\Delta^2_{\rm sphere} - 1)$ where $\epsilon = 0.4$ is chosen to approximate the difference between "continuity" and "JS02" profiles in Smith and Watts [2005].

12.21. Hot Halo Mass Distribution

The algorithm to be used when determining the hot halo mass distribution is selected via the `hotHaloMassDistributionMetho` parameter.

12.21.1. β-profile

Selected with `hotHaloMassDistributionMethod=betaProfile` this method adopts a spherically symmetric β-profile density profile for the hot halo. Specifically,

$$\rho_{\rm hothalo}(r) \propto \left[r^2 + r_{\rm core}^2\right]^{3\beta/2}, \tag{12.164}$$

where the core radius, $r_{\rm core}$, is set using the selected cored profile core radius method (see §12.22). The value of β is specified by the `[hotHaloMassDistributionBeta]` parameter. The profile is normalized such that the current mass in the hot gas profile is contained within the outer radius of the hot halo, $r_{\rm hot,outer}$.

12.21.2. Ricotti (2000)

Selected with `hotHaloMassDistributionMethod=ricotti2000` this method adopts a spherically symmetric β-profile density profile for the hot halo with parameters selected using the fitting function of Ricotti and Shull [2000] who found these by solving for hydrostatic equilibrium in an NFW density profile. Specifically, $\beta = 0.9b$ where

$$b = \frac{2c}{9\Gamma}\left[\log(1+c) - \frac{c}{1+c}\right]^{-1}, \tag{12.165}$$

c is the concentration parameter of the dark matter halo, and Γ is the ratio of virial and gas temperatures, which is assumed to be unity. The core radius is $r_{\rm core} = 0.22 r_{\rm s}$ where $r_{\rm s}$ is the scale radius of the dark matter profile. The profile is normalized such that the current mass in the hot gas profile is contained within the outer radius of the hot halo, $r_{\rm hot,outer}$.

12.21.3. Null

Selected with `hotHaloMassDistributionMethod=null` this method assumes no hot halo mass distribution. It is useful, for example, when performing dark matter-only calculations.

12.21.4. Enzo "hydrostatic"

Selected with `hotHaloMassDistributionMethod=enzoHydrostatic` this method adopts a spherically symmetric density profile for the hot halo motivated by the "hydrostatic" profile available in the ENZO code. Specifically,

$$\rho_{\rm hothalo}(r) \propto \begin{cases} T^{-1} r^{-1} & \text{if } r > r_{\rm core} \\ T^{-1} r_{\rm core}^{-1} & \text{if } r \leq r_{\rm core}, \end{cases} \tag{12.166}$$

where the core radius, $r_{\rm core}$, is set using the selected cored profile core radius method (see §12.22). The profile is normalized such that the current mass in the hot gas profile is contained within the outer radius of the hot halo, $r_{\rm hot,outer}$. Note that the ENZO hydrostatic profile does not include this core, but without introducing this the profile mass can be divergent at small radii.

12.22. Hot Halo Mass Distribution Profile: Cored Profile Core Radius

The algorithm to be used when determining the core radius of cored isothermal hot halo density profiles is selected via the `hotHaloMassDistributionCoreRadiusMethod` parameter.

12.22.1. Growing Core

Selected with `hotHaloMassDistributionCoreRadiusMethod=growingCore`, this method implements a core radius equal to a fraction `[coreRadiusOverScaleRadius]` of the node's dark matter profile scale radius for nodes containing a mass of hot gas equal to the universal baryon fraction times their total mass. For nodes containing less hot gas mass, the core radius is expanded to maintain the same gas density at the virial radius, with a maximum core radius of `[coreRadiusOverVirialRadiusMaximum]` times the node's virial radius.

12.22.2. Virial Fraction

Selected with `hotHaloMassDistributionCoreRadiusMethod=virialRadiusFraction`, this method implements a core radius equal to a fraction `[coreRadiusOverVirialRadius]` of the node's virial radius.

12.23. Hot Halo Ram Pressure Force

The algorithm to be used when determining the ram pressure force due to the hot halo is selected via the `hotHaloRamPressureForceMethod` parameter.

12.23.1. Null

Selected with `hotHaloRamPressureForceMethod=null` this method assumes a zero ram pressure force due to the hot halo.

12.23.2. Font et al. (2008)

Selected with `hotHaloRamPressureForceMethod=Font2008` this method computes the ram pressure force using the algorithm of Font et al. [2008]. Specifically, the ram pressure force is

$$\mathcal{F}_{\rm ram,hot,host} = \rho_{\rm hot,host}(r_{\rm peri}) v^2(r_{\rm peri}), \tag{12.167}$$

where $\rho_{\rm hot,host}(r)$ is the hot halo density profile of the node's host halo, $v(r)$ is the orbital velocity of the node in that host, and $r_{\rm peri}$ is the pericentric radius of the node's orbit.

12.24. Hot Halo Ram Pressure Stripping Radius

The algorithm to be used when determining the radius to which the hot halo is stripped by ram pressure forces is selected via the `hotHaloRamPressureStrippingMethod` parameter.

12.24.1. Virial Radius

Selected with `hotHaloRamPressureStrippingMethod=virialRadius` this method sets the ram pressure stripping radius equal to the virial radius of the halo. The effectively results in no ram pressure stripping.

12.24.2. Font et al. (2008)

Selected with `hotHaloRamPressureStrippingMethod=Font2008` this method computes the ram pressure stripping radius using the algorithm of Font et al. [2008]. Specifically, the radius, $r_{\rm rp}$, is computed as the solution of

$$\alpha_{\rm rp} \frac{G M_{\rm satellite}(r_{\rm rp}) \rho_{\rm hot,satellite}(r_{\rm rp})}{r_{\rm rp}} = \mathcal{F}_{\rm ram,hot,host}, \tag{12.168}$$

where $M_{\rm satellite}(r)$ is the total mass of the satellite within radius r, $\mathcal{F}_{\rm ram,hot,host}$ is the ram pressure force due to the hot halo (computed using the selected hot halo ram pressure force method; see §12.23). The parameter $\alpha_{\rm rp}$ =`[ramPressureStrippingFormFactor]` is a geometric factor of order unity.

12.25. Hot Halo Ram Pressure Stripping Timescale

The algorithm to be used when determining the timescale on which the hot halo is stripped by ram pressure forces is selected via the `hotHaloRamPressureStrippingTimescaleMethod` parameter.

12.25.1. Halo Dynamical Timescale

Selected with `hotHaloRamPressureStrippingTimescaleMethod=haloDynamicalTime` this method sets the ram pressure stripping timecale equal to the dynamical time of the associated halo.

12.25.2. Ram Pressure Acceleration

Selected with `hotHaloRamPressureStrippingTimescaleMethod=ramPressureAcceleration` this method computes the ram pressure stripping timecale from the acceleration imparted by the ram pressure force. Following Roediger and BrÃČÅŠggen [2007] this is approximated as:

$$a_{\rm rampressure} = P_{\rm rampressure}/\Sigma, \tag{12.169}$$

where $P_{\rm rampressure}$ is the ram pressure force per unit area, and Σ is the surface density of gas. The associated timescale to accelerate gas over a distance $r_{\rm outer}$ (the current outer radius of the hot halo) is then:

$$\tau_{\rm rampressure} = \sqrt{2 r_{\rm outer} \Sigma_{\rm outer} / P_{\rm rampressure}}. \tag{12.170}$$

12.26. Hot Halo Outflow Reincorporation

The algorithm to be used when determining the rate at which outflowed gas is reincorporated into the hot halo is selected via the `hotHaloOutflowReincorporationMethod` parameter.

12.26.1. Velocity Maximum Scaling

Selected with `hotHaloOutflowReincorporationMethod=hotHaloOutflowReincorporationVelocityMaximumScaling` this method assumes a reincorporation rate of:

$$\dot{M}_{\rm reincorporation} = M_{\rm outflowed}/t_{\rm reincorporation}, \tag{12.171}$$

where

$$t_{\rm reincorporation} = \tau_{\rm reincorporation} \left(\frac{V_{\rm max}}{200 {\rm km/s}} \right)^{\alpha_{\rm reincorporation}} (1+z)^{\beta_{\rm reincorporation}}, \tag{12.172}$$

where $\tau_{\rm reincorporation}$ =`[hotHaloOutflowReincorporationVlctyMxSclngTimeScale]`, $\alpha_{\rm reincorporation}$ =`[hotHaloOutflowRei` and $\beta_{\rm reincorporation}$ =`[hotHaloOutflowReincorporationVlctyMxSclngRedshiftExponent]`.

12.27. Hot Halo Temperature Profile

The algorithm to be used when determining the hot halo temperature profile is selected via the `hotHaloTemperatureMethod` parameter.

12.27.1. Virial Temperature

Selected with `hotHaloTemperatureMethod=virial` this method assumes an isothermal halo with a temperature equal to the virial temperature of the halo.

12.27.2. Enzo "Hydrostatic"

Selected with `hotHaloTemperatureMethod=enzoHydrostatic` this method assumes the "hydrostatic" temperature profile available in the ENZO code. Specifically,

$$T(r) = \max\left(\frac{\mathrm{G}M(<r)\mu m_\mathrm{H}}{3\mathrm{k_B}r}, T_\mathrm{min}\right), \tag{12.173}$$

where $M(<r)$ is the total mass enclosed within radius r, μ is the primordial mean atomic mass, and T_min =`[enzoHydrostaticTemperatureMinimum]` is a temperature floor introduced so as to avoid the temperature reaching abitrarily low masses.

12.28. Initial Mass Functions

The stellar IMF subsystem supports multiple IMFs and extensible algorithms to select which IMF to use based on the physical conditions of star formation.

12.28.1. Initial Mass Function Selection

The method to use for selecting which IMF to use is specified by the `imfSelectionMethod` parameter.

Fixed

Selected by `imfSelectionMethod=fixed`, this method uses a fixed IMF irrespective of physical conditions. The IMF to use is specified by the `imfSelectionFixed` parameter (e.g. setting this parameter to `Salpeter` selects the Salpeter IMF).

Disk and Spheroid

Selected by `imfSelectionMethod=diskSpheroid`, this method uses different IMFs for star formation in disks and spheroids irrespective of other physical conditions. The IMFs to use are specified by the `imfSelectionDisk` and `imfSelectionSpheroid` parameters (e.g. setting one of these parameters to `Salpeter` selects the Salpeter IMF).

12.28.2. Initial Mass Functions

A variety of different IMF s are available. Each IMF registers itself with the Galacticus IMF subsystem and can then be looked-up by name or internal index. All IMFs are assumed to be continuous in M, unless otherwise noted and normalized to unit mass. Each IMF supplies a recycled fraction and metal yield for use in the instantaneous recycling approximation. These can be set via the parameters `imf[imfName]RecycledInstantaneous` and `imf[imfName]YieldInstantaneous` where `[imfName]` is the name of the IMF. Their default values were computed using Galacticus's internal stellar astrophysics modules for a Solar metallicity population with age of 13.8 Gyr.

Baugh et al. (2005) Top-Heavy

The `Baugh2005TopHeavy` IMF is defined by [Baugh et al., 2005]:

$$\phi(M) \propto M^{-1} \text{ for } 0.15M_\odot < M < 125M_\odot \tag{12.174}$$

BPASS

The `BPASS` IMF is defined by:

$$\phi(M) \propto \begin{cases} M^{-1.30} & \text{for } 0.1M_\odot < M < 0.5M_\odot \\ M^{-2.35} & \text{for } 1M_\odot < M < 120M_\odot \\ 0 & \text{otherwise.} \end{cases} \tag{12.175}$$

Chabrier

The `Chabrier` IMF is defined by [Chabrier, 2001]:

$$\phi(M) \propto \begin{cases} M^{-1} \exp(-[\log_{10}(M/M_\mathrm{c})/\sigma_\mathrm{c}]^2/2) & \text{for } M_\mathrm{l} < M < M_\mathrm{t} \\ M^{\alpha} & \text{for } M_\mathrm{t} < M < M_\mathrm{u} \\ 0 & \text{otherwise,} \end{cases} \tag{12.176}$$

where $\sigma_\mathrm{c} =$`[imfChabrierSigma]`, $M_\mathrm{c} =$`[imfChabrierMass]`$M_\odot$, $\alpha =$`[imfChabrierExponent]`, $M_\mathrm{t} =$`[imfChabrierMassTran`
$M_\mathrm{l} =$`[imfChabrierMassLower]`$M_\odot$, and $M_\mathrm{u} =$`[imfChabrierMassUpper]`$M_\odot$.

Kennicutt

The `Kennicutt` IMF is defined by [Kennicutt, 1983]:

$$\phi(M) \propto \begin{cases} M^{-1.25} & \text{for } 0.10M_\odot < M < 1.00M_\odot \\ M^{-2.00} & \text{for } 1.00M_\odot < M < 2.00M_\odot \\ M^{-2.30} & \text{for } 2.00M_\odot < M < 125M_\odot \\ 0 & \text{otherwise.} \end{cases} \tag{12.177}$$

Kroupa

The `Kroupa` IMF is defined by [Kroupa, 2001]:

$$\phi(M) \propto \begin{cases} M^{-0.3} & \text{for } 0.01M_\odot < M < 0.08M_\odot \\ M^{-1.8} & \text{for } 0.08M_\odot < M < 0.5M_\odot \\ M^{-2.7} & \text{for } 0.5M_\odot < M < 1M_\odot \\ M^{-2.3} & \text{for } 1M_\odot < M < 125M_\odot \\ 0 & \text{otherwise.} \end{cases} \tag{12.178}$$

Miller-Scalo

The `Miller-Scalo` IMF is defined by [Miller and Scalo, 1979]:

$$\phi(M) \propto \begin{cases} M^{-1.25} & \text{for } 0.10M_\odot < M < 1.00M_\odot \\ M^{-2.00} & \text{for } 1.00M_\odot < M < 2.00M_\odot \\ M^{-2.30} & \text{for } 2.00M_\odot < M < 10.0M_\odot \\ M^{-3.30} & \text{for } 10.0M_\odot < M < 125M_\odot \\ 0 & \text{otherwise.} \end{cases} \tag{12.179}$$

Piecewise Power-law

Arbitrary piecewise power-law IMFs can be defined using the `PiecewisePowerLaw` method. The IMF will be constructed such that:

$$\phi(M) \propto M^{\alpha_i} \text{ if } M_i \leq M < M_{i+1}, \tag{12.180}$$

where $i = 1 \ldots N$, the M_i are given by `[imfPiecewisePowerLawMassPoints]` and the α_i are given by `[imfPiecewisePowerLawExponents]`. (Note that `[imfPiecewisePowerLawMassPoints]` must contain $N+1$ elements, while `[imfPiecewisePowerLawExponents]` contains only N elements.) The normalization of each power-law piece is chosen to ensure a continuous IMF that is normalized to unit mass overall.

Salpeter

The `Salpeter` IMF is defined by [Salpeter, 1955]:

$$\phi(M) \propto \begin{cases} M^{-2.35} & \text{for } 0.1M_\odot < M < 125M_\odot \\ 0 & \text{otherwise.} \end{cases} \tag{12.181}$$

Scalo

The `Scalo` IMF is defined by [Scalo, 1986]:

$$\phi(M) \propto \begin{cases} M^{+1.60} & \text{for } 0.10M_\odot < M < 0.18M_\odot \\ M^{-1.01} & \text{for } 0.18M_\odot < M < 0.42M_\odot \\ M^{-2.75} & \text{for } 0.42M_\odot < M < 0.62M_\odot \\ M^{-2.08} & \text{for } 0.62M_\odot < M < 1.18M_\odot \\ M^{-3.50} & \text{for } 1.18M_\odot < M < 3.50M_\odot \\ M^{-2.63} & \text{for } 3.50M_\odot < M < 125M_\odot \\ 0 & \text{otherwise.} \end{cases} \tag{12.182}$$

12.29. Intergalactic Medium State

The thermal and ionization state of the intergalactic medium is implemented by the algorithm selected using the `intergalaticMediumStateMethod` parameter.

12.29.1. Simple

Selected using `intergalaticMediumStateMethod=simple`, this method provides a simple model of reionization in which the universe is assumed to be fully neutral prior to the redshift given by `[igmStateSimpleReionizationRedsh` and fully ionized thereafter. The temperature is given by `[igmStateSimplePreReionizationTemperature]` before reionization, and `[igmStateSimpleReionizationTemperature]` thereafter.

12.29.2. RecFast

Selected using `intergalaticMediumStateMethod=RecFast`, this method computes the state of the intergalactic medium using the RecFast code Seager et al. [2000], Wong et al. [2008]. The RecFast code will be downloaded and run to compute the intergalactic medium state as needed, which will then be stored for future use.

12.29.3. File

Selected using `intergalaticMediumStateMethod=file`, this method reads the state of the intergalactic medium from a file and interpolates in the tabulated results. The HDF5 file containing the table should have the following form:

```
HDF5 "igmState.hdf5" {
GROUP "/" {
   ATTRIBUTE "fileFormat" {
      DATATYPE  H5T_IEEE_F64LE
      DATASPACE  SCALAR
      DATA {
      (0): 1
      }
   }
   GROUP "Parameters" {
      ATTRIBUTE "HubbleConstant" {
         DATATYPE  H5T_IEEE_F64LE
         DATASPACE  SCALAR
         DATA {
         (0): 67.8
         }
      }
      ATTRIBUTE "OmegaBaryon" {
         DATATYPE  H5T_IEEE_F64LE
         DATASPACE  SCALAR
         DATA {
         (0): 0.0484
         }
      }
      ATTRIBUTE "OmegaDarkEnergy" {
         DATATYPE  H5T_IEEE_F64LE
         DATASPACE  SCALAR
         DATA {
         (0): 0.692
         }
      }
      ATTRIBUTE "OmegaMatter" {
         DATATYPE  H5T_IEEE_F64LE
         DATASPACE  SCALAR
         DATA {
         (0): 0.308
         }
      }
      ATTRIBUTE "Y_He" {
         DATATYPE  H5T_IEEE_F64LE
         DATASPACE  SCALAR
         DATA {
         (0): 0.22
         }
      }
```

```
      ATTRIBUTE "temperatureCMB" {
         DATATYPE  H5T_IEEE_F64LE
         DATASPACE  SCALAR
         DATA {
         (0): 2.725
         }
      }
   }
   DATASET "electronFraction" {
      DATATYPE  H5T_IEEE_F64LE
      DATASPACE  SIMPLE { ( 10000 ) / ( 10000 ) }
   }
   DATASET "hIonizedFraction" {
      DATATYPE  H5T_IEEE_F64LE
      DATASPACE  SIMPLE { ( 10000 ) / ( 10000 ) }
   }
   DATASET "heIonizedFraction" {
      DATATYPE  H5T_IEEE_F64LE
      DATASPACE  SIMPLE { ( 10000 ) / ( 10000 ) }
   }
   DATASET "matterTemperature" {
      DATATYPE  H5T_IEEE_F64LE
      DATASPACE  SIMPLE { ( 10000 ) / ( 10000 ) }
      ATTRIBUTE "units" {
         DATATYPE  H5T_STRING {
            STRSIZE 6;
            STRPAD H5T_STR_NULLTERM;
            CSET H5T_CSET_ASCII;
            CTYPE H5T_C_S1;
         }
         DATASPACE  SCALAR
         DATA {
         (0): "Kelvin"
         }
      }
      ATTRIBUTE "unitsInSI" {
         DATATYPE  H5T_IEEE_F64LE
         DATASPACE  SCALAR
         DATA {
         (0): 1
         }
      }
   }
   DATASET "redshift" {
      DATATYPE  H5T_IEEE_F64LE
      DATASPACE  SIMPLE { ( 10000 ) / ( 10000 ) }
   }
}
}
```

The `electronFraction`, `hIonizedFraction`, `heIonizedFraction`, and `matterTemperature` datasets contain the relevaqnt quantity for each redshift in the `redshift` dataset.

12.30. Chemical State

The chemical state of gas is implemented by the algorithm selected using the `ionizatonStateMethod` parameter.

12.30.1. Atomic Collisional Ionization Equilibrium Using CLOUDY

Selected using `ionizatonStateMethod=atomic_CIE_Cloudy`, this method computes the chemical state using the CLOUDY code and under the assumption of collisional ionization equilibrium with no molecular contribution. Abundances are Solar, except for zero metallicity calculations which use CLOUDY's "primordial" metallicity. The helium abundance for non-zero metallicity is scaled between primordial and Solar values linearly with metallicity. The CLOUDY code will be downloaded and run to compute the cooling function as needed, which will then be stored for future use. As this process is slow, a precomputed table is provided with GALACTICUS. If metallicities outside the range tabulated in this file are required it will be regenerated with an appropriate range.

12.30.2. Collisional Ionization Equilibruim From File

Selected using `ionizatonStateMethod=CIE_from_file`, in this method the chemical state is read from a file specified by the `chemicalStateFile` parameter. The format of this file is specified in §16.4.1. The chemical state is assumed to be computed under conditions of collisional ionization equilibrium and therefore densities scale as ρ. Optional HI and HII densities, if present in the file, will be read and used when returning the densities of "chemical" species.

12.31. Mass Function Incompleteness

The incompleteness of observed mass functions is modeled using the method specified by the `massFunctionIncompletenessMe` parameter.

12.31.1. Complete

Selected with `massFunctionIncompletenessMethod=complete`, this method assumes a fully complete mass function.

12.31.2. Surface brightness

Selected with `massFunctionIncompletenessMethod=surfaceBrightness`, this method models the surface brightness distribution of galaxies as a normal distribution with mean $\langle\mu\rangle(M) = \alpha \log_{10}(M/M_0) + \beta$, with root-variance σ, where $\alpha =$ `[massFunctionIncompletenessSurfaceBrightnessModelSlope]`, $\beta =$ `[massFunctionIncompletenessSurfaceBrightnessZeroPoint]`, and $\sigma =$ `[massFunctionIncompletenessSurfaceBrigh` The completeness is the fraction of this distribution above the surface brightness limit given by `[massFunctionIncompleteness`

12.32. Merger Tree Construction

Merger trees are "constructed[5]" by the method specified by the `mergerTreeConstructMethod` parameter.

12.32.1. Read From File

Selected with `mergerTreeConstructMethod=read`, this method reads merger tree structures from file by the designated importer (see §12.37).

12.32.2. Build

Selected with `mergerTreeConstructMethod=build`, this method first creates a distribution of tree root halo masses and then builds a merger tree using the algorithm specified by the `mergerTreeBuildMethod` parameter.

If `[mergerTreeBuildTreesHaloMassDistribution]=read`, then these masses will be read from a file specified by `[mergerTreeBuildTreeMassesFile]`. Otherwise, the root halo masses are selected to range between `mergerTreeBuildHaloMassMinimum` and `mergerTreeBuildHaloMassMaximum` with `mergerTreeBuildTreesPerDecad` trees per decade of root halo mass on average. Trees are rooted at `mergerTreeBuildTreesBaseRedshift` and tree building will begin with the `mergerTreeBuildTreesBeginAtTree`$^{\rm th}$ tree[6]. The root halo masses are usually drawn from the density function specified by the halo mass function sampling density method (see §12.11.6). The distribution of x is determined by the input parameter `mergerTreeBuildTreesHaloMassDistribution` with options:

`uniform` x is distributed uniformly between 0 and 1;

`quasi` x is distributed using a quasi-random sequence;

`random` x is distributed using a pseudo-random sequence;

`read` halos masses are instead read from a file.

In the case of reading root halo masses from a file, the file can be either an XML or HDF5 file. An XML file should have the following form:

```
<mergerTrees>
 <treeRootMass>13522377303.5998</treeRootMass>
 <treeRootMass>19579530191.8709</treeRootMass>
 <treeRootMass>21061025282.9613</treeRootMass>
 .
 .
 .
 <treeWeight>13522377303.5998</treeWeight>
 <treeWeight>19579530191.8709</treeWeight>
 <treeWeight>21061025282.9613</treeWeight>
 .
 .
 .
</mergerTrees>
```

[5] By "construct" we mean any process of creating a representation of a merger tree within Galacticus.

[6] This will normally be set to 1 to begin with the first tree. Other values allow to begin on later trees for debugging purposes.

where each `treeRootMass` element gives the mass (in Solar masses) of the root halo of a tree to generate, and the (optional) `treeWeight` elements give the corresponding weight (in units of Mpc^{-3}) to assign to each tree. An HDF5 file should have the following structure:

```
HDF5 {
GROUP '/' {
   DATASET 'treeRootMass' {
      DATATYPE  H5T_IEEE_F64BE
      DATASPACE  SIMPLE { ( * ) / ( * ) }
   }
   DATASET 'treeWeight' {
      DATATYPE  H5T_IEEE_F64BE
      DATASPACE  SIMPLE { ( * ) / ( * ) }
   }
}
}
```

where the `treeRootMass` dataset contains the mass (in Solar masses) of the root halo of a tree to generate, and the (optional) `treeWeight` dataset contains the weight (in units of Mpc^{-3}) to assign to each tree.

12.32.3. Fully Specified

Selected with `mergerTreeConstructMethod=fullySpecified`, this method will construct a merger tree, and set properties of components in each node, using a description read from an XML document. The document is specified via the `[mergerTreeConstructFullySpecifiedFileName]` input parameter.

The tree specification document looks as follows:

```
<!-- Simple initial conditions test case -->
<initialConditions>

  <node>
    <index>2</index>
    <parent>1</parent>
    <firstChild>-1</firstChild>
    <sibling>-1</sibling>
    <basic>
      <time>1.0</time>
      <timeLastIsolated>1.0</timeLastIsolated>
      <mass>1.0e12</mass>
      <accretionRate>7.9365079e9</accretionRate>
    </basic>
    <spin>
      <spin>0.1</spin>
    </spin>
    <disk>
      <massGas>1.0e10</massGas>
      <angularMomentum>1.0e10</angularMomentum>
      <abundancesGas>
<metals>1.0e9</metals>
<Fe>1.0e9</Fe>
      </abundancesGas>
```

```
    </disk>
  </node>

  <node>
    <index>1</index>
    <parent>-1</parent>
    <firstChild>2</firstChild>
    <sibling>-1</sibling>
    <basic>
      <time>13.8</time>
      <timeLastIsolated>13.8</timeLastIsolated>
      <mass>1.1e12</mass>
      <accretionRate>7.8125e9</accretionRate>
    </basic>
    <position>
      <position>1.23</position>
      <position>6.31</position>
      <position>3.59</position>
    </position>
  </node>

</initialConditions>
```

The document consists of a set of `node` elements, each of which defines a single node in the merger tree. Each `node` element must specify the `index` of the node, along with the index of the node's `parent`, `firstChild`, and `sibling`.

Each `node` element may contain elements which specify the properties of a component in the node. For example, a `basic` element will specify properties of the "basic" component. If multiple elements for a given component type are present, then multiple instances of that component will be created in the node.

Within a component definition element scalar properties are set using an element with the same name as that property (e.g. `mass` in the `basic` components in the above example). Rank-1 properties are set using a list of elements with the same name as the property (e.g. `position` in the `position` component in the above example).

For composite properties (e.g. abundances), the specification element should contain sub-elements that specify each property of the composite. Currently only the `abundances` object supports specification in this way, as detailed below:

`abundances` (See `abundancesGas` in the above example.) The total metal content is specified via a `metals` element. If other elements are being tracked, their content is specified via an element with the short-name of the element (e.g. `Fe` for iron).

12.32.4. Smooth Accretion

Selected with `mergerTreeConstructMethod=smoothAccretion`, this method builds a branchless merger tree with a smooth accretion history using the selected mass accretion history method (see §16.4.1). The tree has a final mass of `mergerTreeHaloMass` (in units of $M_\odot$) at redshift `mergerTreeBaseRedshift` and is continued back in time by decreasing the halo mass by a factor `mergerTreeHaloMassDeclineFactor` at each new node until a specified `mergerTreeHaloMassResolution` (in units of $M_\odot$) is reached.

12.32.5. State Restore

Selected with `mergerTreeConstructMethod=stateRestore`, this method will restore a merger tree whose complete internal state was written to file. It is intended primarily for debugging purposes to allow a tree to begin processing just prior to the point of failure. To use this method, the following procedure should be followed:

1. Identify a point in the evolution of the tree suitably close to, but before, the point of failure;
2. Insert approrpiate code into Galacticus to have it call the function to store the state of the file and then stop, e.g.:

```
use Merger_Trees_State_Store
.
.
.
if (<conditions are met>) then
   call Merger_Tree_State_Store(thisTree,'storedTree.dat')
   stop 'tree internal state was stored'
end if
```

3. Run the model ensuring that `[stateFileRoot]` is set to a suitable file root name to allow the internal state of Galacticus to be stored;
4. Remove the code inserted above and recompile;
5. Run Galacticus with an input parameter file identical to the one used previously except with `[mergerTreeConstructMethod]=stateRestore`, `[stateFileRoot]` removed, `[stateRetrieveFileRoot]` set to the value previously used for `[stateFileRoot]` and `[mergerTreeStateStoreFile]=storedTree.dat`.

This should restore the tree and the internal state of Galacticus precisely from the point where they were saved and produce the same subsequent evolution.

Note that currently this method does not support storing and restoring of trees which contain components that have more than one instance.

12.33. Merger Tree Building Mass Resolution

The method to be used for computing the mass resolution to use when building merger trees is specified by the `mergerTreesBuildMassResolutionMetod` parameter.

12.33.1. Fixed

Selected with `mergerTreesBuildMassResolutionMetod=fixed`, this method assumes a fixed mass resolution of `[mergerTreeBuildMassResolutionFixed]` for all merger trees.

12.33.2. Scaled

Selected with `mergerTreesBuildMassResolutionMetod=scaled`, this method computes the mass resolution to be the larger of `[mergerTreeBuildMassResolutionScaledMinimum]` and `[mergerTreeBuildMassResolutionScaledF` where $M_{\rm base}$ is the base mass of the merger tree.

12.34. Merger Tree Branching

The method to be used for computing branching probabilities in merger trees is specified by the `treeBranchingMethod` parameter.

12.34.1. Modified Press-Schechter

Selected with `treeBranchingMethod=modifiedPress-Schechter`, this method uses the algorithm of Parkinson et al. [2008] to compute branching ratios. The parameters G_0, γ_1 and γ_2 of their algorithm are specified by the input parameters `modifiedPressSchechterG0`, `modifiedPressSchechterGamma1` and `modifiedPressSchechterGamma2` respectively. Additionally, the parameter `modifiedPressSchechterFirstOrderAccuracy` limits the step in $\delta_{\rm crit}$ so that it never exceeds `modifiedPressSchechterFirstOrderAccuracy` $\sqrt{2[\sigma^2(M_2/2)-\sigma^2(M_2)]}$, which ensures the the first order expansion of the merging rate that is assumed is accurate. To find bounds on the branching probability, we make use of the fact that eqn. (4) of Parkinson et al. [2008] can be written as

$$\frac{{\rm d}f}{{\rm d}t} = \frac{{\rm d}t}{{\rm d}\omega}\int_{M_{\rm min}}^{M/2} \frac{M}{M'}\frac{{\rm d}f}{{\rm d}t}\frac{{\rm d}S}{{\rm d}M'}\left|\frac{{\rm d}t}{{\rm d}\omega}\right| G[\omega,\sigma(M),\sigma(M')]{\rm d}M'. \tag{12.183}$$

By holding the M' in the denominator of the first term in the integrand fixed, we obtain an analytic solution to the integral in terms of hypergeometric functions. If we fix this M' at $M_{\rm min}$ we obtain an upper limit on the branching probability, while if we fix it to $M/2$ a lower limit is obtained.

Calculation of branching probabilities involves computation of several hypergeometric functions which are numerically slow. Two parameters control the accuracy and application of these functions. First, `[modifiedPressSchechterHypergeometricPrecision]` ($=10^{-6}$) specifies the fractional tolerance to which these functions should be computed. Second, if `[modifiedPressSchechterTabulateHypergeometricFactors]=true` then these functions will be tabulated for rapid lookup (at some loss of precision).

12.34.2. Generalized Press-Schechter

Selected with `treeBranchingMethod=generalizedPress-Schechter`, this method computes branching probabilities from solutions to the excursion set barrier first crossing rate problem (using the selected `excursionSetFirstCrossingMethod`; see §16.4.1). Specifically, the branching probability per unit time is:

$$\frac{{\rm d}f}{{\rm d}t} = \frac{{\rm d}t}{{\rm d}\omega}\int_{M_{\rm min}}^{M/2} \frac{M}{M'}\frac{{\rm d}f}{{\rm d}t}\frac{{\rm d}S}{{\rm d}M'}\left|\frac{{\rm d}t}{{\rm d}\omega}\right| G[\omega,\sigma(M),\sigma(M')]{\rm d}M', \tag{12.184}$$

where $\omega=\delta_{\rm c,0}/D(t)$. The rate of accretion of mass in halos below the resolution limit of the merger tree is

$$\frac{{\rm d}R}{{\rm d}t} = \frac{{\rm d}t}{{\rm d}\omega}\int_{0}^{M_{\rm min}} \frac{{\rm d}f}{{\rm d}t}\frac{{\rm d}S}{{\rm d}M'}\left|\frac{{\rm d}t}{{\rm d}\omega}\right| G[\omega,\sigma(M),\sigma(M')]{\rm d}M'. \tag{12.185}$$

In the above, $G[\omega,\sigma(M),\sigma(M')]$ is a modification to the merger rate as computed by the selected `treeBranchingModifierMethod` (see §16.4.1). If `[generalizedPressSchechterSmoothAccretion]=true` then smooth accretion (i.e. accretion of matter not in dark matter halos) is accounted for at the rate:

$$\frac{{\rm d}R_{\rm s}}{{\rm d}t} = \frac{{\rm d}t}{{\rm d}\omega}G[\omega,\sigma_{\rm max},\sigma(M')]\frac{{\rm d}\,\widetilde{f}}{{\rm d}t}, \tag{12.186}$$

where $\sigma_{\rm max}$ is the peak value of $\sigma(M)$ (for the lowest mass halos) and ${\rm d}\,\widetilde{f}\,/{\rm d}t$ is the rate at which excursion set trajectories *fail* to cross the barrier on any mass scale.

12.35. Merger Tree Branching Modifier

The method to be used for modifying branching probabilities in merger trees is specified by the `treeBranchingModifierMetho` parameter.

12.35.1. Null

Selected with `treeBranchingModifierMethod=null`, this method makes no change to the branching probability.

12.35.2. Parkinson, Cole & Helly (2008)

Selected with `treeBranchingModifierMethod=Parkinson-Cole-Helly2008`, this method modifies branching probabilities according to the prescription of Parkinson et al. [2008]. Specifically, the branching probability is multiplied by:

$$G(\delta_\mathrm{p},\sigma_\mathrm{c},\sigma_\mathrm{p}) = G_0 \left(\frac{\sigma_\mathrm{p}}{\sigma_\mathrm{p}}\right)^{\gamma_1} \left(\frac{\delta_\mathrm{p}}{\sigma_\mathrm{p}}\right)^{\gamma_2} \tag{12.187}$$

where δ_p is the current critical overdensity for collapse for the parent halo, and σ_c and σ_p are the root-variance of the smooth mass-density field on scales corresponding to the masses of child and parent halos respectively. The parameters of the fit can be adjusted via input parameters: $G_0 =$`[modifiedPressSchechterG0]`, $\gamma_1 =$`[modifiedPressSchechterGamma1]`, and $\gamma_2 =$`[modifiedPressSchechterGamma2]`.

12.36. Merger Tree Building

The method to be used for building merger trees is specified by the `mergerTreeBuildMethod` parameter.

12.36.1. Cole et al. (2000) Algorithm

Selected with `mergerTreeBuildMethod=Cole2000`, this method uses the algorithm described by Cole et al. [2000] (with minor modifications described below), with a branching probability method selected via the `treeBranchingMethod` parameter. This action of this algorithm is controlled by the following parameters:

`mergerTreeBuildCole2000MergeProbability` The maximum probability for a binary merger allowed in a single timestep. This allows the probability to be kept small, such the the probability for multiple mergers within a single timestep is small.

`mergerTreeBuildCole2000AccretionLimit` The maximum fractional change in mass due to sub-esolution accretion allowed in any given timestep when building the tree.

`mergerTreeBuildCole2000HighestRedshift` The highest redshift to which the tree should be built. Any branch reaching this redshift will be terminated. Typically this should be set to a high value such that branches terminate when the resolution limit it reached, but specifying a maximum redshift can be useful in some situations.

`modifiedPressSchechterUseCDMAssumptions` Stronger bounds on the branching probability can be found if certain assumptions[7], which are valid for CDM but not necessarily for other models, are made. Setting this option to `true` allows these assumptions to be used (and will speed up the algorithm).

[7] Specifically, $\alpha(=-\mathrm{d}\log\sigma/\mathrm{d}\log M) > 0$ and $\mathrm{d}\alpha/\mathrm{d}M > 0$ [Parkinson et al., 2008].

`mmergerTreeBuildCole2000BranchIntervalStep` If `true`, instead of limiting each time step such that the probability of branching is less than `mergerTreeBuildCole2000MergeProbability`, the interval to the next branching event will be drawn from a negative exponential with the appropriate rate. If this exceeds the maximum allowed timestep based on other considerations (e.g. the accretion limit), no branching occurs, and the timestep proceeds[8]. If the interval is less than the maximum allowed timestep, branching occurs at that point. In the regime of high branching rates (which occur when the branch being grown is far above the mass resolution), this approach allows for larger timesteps to be taken.

The minimum halo mass that the algorithm will follow is determined by the selection merger tree building mass resolution method (see §12.33). Mass accretion below this scale is treated as smooth accretion and branches are truncated once they fall below this mass.

Modifications

In the original Cole et al. [2000], when a branch split occurred masses, M_2 and M_3, of the two new halos were selected by first drawing the mass M_2 from the branching distribution function in the range $M_{\rm res}$ to $M_1/2$ (where M_1 is the mass of the parent halo, and $M_{\rm res}$ is the mass resolution being used for the tree), and then setting

$$M_3 = M_1(1-F) - M_2 \tag{12.188}$$

where F is the fraction of the parent halo mass gained through sub-resolution accretion in this timestep. As the sub-resolution accretion is removed entirely from the mass M_3 and not from M_2 this can lead to an assymetry in progenitor mass functions close to $M_1/2$. Therefore, we instead set the progenitor masses by first drawing a mass M_2' from the mass branching distribution function and then setting

$$\begin{aligned} M_2 &= M_2'(1-F), \\ M_3 &= (M_1 - M_2')(1-F), \end{aligned} \tag{12.189}$$

which ensures a symmetric treatment of subresolution accretion close to $M_1/2$.

12.37. Merger Tree Importing

When merger trees are to be read from file, a number of different file formats are supported. An "importer" class is used to read these files and place the contents into internal data structures that Galacticus can then manipulate. The importer class to use is specifide by the `[mergerTreeImporterMethod]` parameter.

12.37.1. Galacticus

Selected with `mergerTreeImporterMethod=galacticus`, this method reads merger tree structures from an HDF5 file specified by the `mergerTreeReadFileName` parameter. The structure of these HDF5 files is described in §5.2.

12.37.2. Sussing Merger Trees

Selected with `mergerTreeImporterMethod=sussing`, this method reads merger tree structures from files following the format designed for the "Sussing Merger Trees" workshop [Srisawat et al., 2013], along

[8] Note that we do not have to concern ourselves in the subsequent timestep with the fact that no branching occurred in the previous timestep because of the memorylessness nature of the negative exponential distribution. That is, the distribution of branching intervals conditioned on the fact that no branching occurred in the previous timestep, is just the same negative exponential distribution.

with Amiga's Halo Finder (AHF) format halo catalogs. A descriptor file must be specified via the `[mergerTreeReadFileName]` parameter. This descriptor file should have the following format:

```
simulation.txt
MergerTree+AHF.txt
snapidzred.txt
AHF/62.5_dm_000.z50.000.AHF_halos
AHF/62.5_dm_001.z30.000.AHF_halos
AHF/62.5_dm_002.z19.916.AHF_halos
.
.
.
AHF/62.5_dm_061.z0.000.AHF_halos
```

in which each line specifies a file to be read (by default path names are relative to the location of the descriptor file—fully-qualified path names can also be given).

The first line identifies a file which specifies properties of the simulation. This file should look like:

```
WMAP7 cosmology:
----------------
Omega0          =       0.272
OmegaLambda0    =       0.728
h               =       0.704

simulation:
-----------
B               =       62.5 Mpc/h
N               =       270^3 particles
```

Currently only the cosmological parameter and box length are read from this file.

The second line identifies the merger tree file which must be in the format specified by Srisawat et al. [2013].

The third line of the descriptor file specifies a snapshot file which should have the following format:

```
#    snapnum       a             z          t(t0)      t(year)
           0    0.0196080      49.9996   0.00354284  4.87485e+07
           1    0.0322580      30.0001   0.00747572  1.02864e+08
           2    0.0478110      19.9157    0.0134888  1.85602e+08
           3    0.0519650      18.2437    0.0152842  2.10306e+08
           4    0.0564190      16.7245    0.0172905  2.37912e+08
           5    0.0611880      15.3431    0.0195280  2.68700e+08
           6    0.0662870      14.0859    0.0220186  3.02969e+08
             .
             .
             .
```

This file must contain one line for each snapshot of the simulation, giving the snapshot number, expansion factor, redshift, fractional time (relative to present day), and age of the universe (in years).

Subsequent lines identify the AHF halo files for each snapshot (files can be listed in any order).

Merger tree files of this type can be split into subvolumes before processing. This is useful if the file is too large to read into memory in one go. The number of subvolumes to use (in each of the three dimensions of the simulation cube) is specified by the `[mergerTreeImportSussingSubvolumeCount]` parameter. The

specific subvolume to process is specified by the [mergerTreeImportSussingSubvolumeIndex] parameter, which should give the index (running from 0 to [mergerTreeImportSussingSubvolumeCount] -1) in each dimension (whitespace separated). To ensure that no halos are missed from trees near the edge of the subvolume, a buffer region around the subvolume is also read. The width of this buffer (in units of Mpc$/h$ to follow the format convention) is specified via the [mergerTreeImportSussingSubvolumeBuffer] parameter.

12.38. Merger Tree Pre-evolution Processing

Arbitrary processing of merger trees prior to their evolution can be carried out using the mergerTreePreEvolveTask directive (see §16.4.3). Currently defined tasks are defined below.

12.38.1. Enforce Monotonic Mass Growth

This task enforces monotonic growth a halo mass along each branch of each merger tree. It does this by searching the tree for nodes which are less massive than the sum of the masses of their immediate progenitors, and increasing the mass of such nodes to equal the sum of the masses of their immediate progenitors. To enforce monotonic mass growth along branches set [mergerTreeEnforceMonotonicGrowth]=true.

12.38.2. Interpolate Tree to Time Grid

This task will interpolate the merger tree structure onto a new array of timesteps if [mergerTreeRegridTimes]=true. The timestep array is specified via the parameters:

[mergerTreeRegridSpacing] The spacing of the timesteps. Five options are available: linear will space timesteps uniformly in expansion factor, logarithmic will space timesteps uniformly in the logarithm of expansion factor, logCriticalDensity will space timesteps uniformly in the logarithm of critical density, δ_c, millennium will use times corresponding to the redshifts of snapshots in the Millennium Simulation database, while read will use times corresponding to the redshifts specified in the mergerTreeRegridRedshifts parameter;

[mergerTreeRegridStartExpansionFactor] The smallest expansion factor in the array (ignored for millennium and read spacings);

[mergerTreeRegridEndExpansionFactor] The largest expansion factor in the array (ignored for millennium and read spacings);

[mergerTreeRegridCount] The number of timesteps in the array;

Along each branch of the tree, new halos are inserted at times corresponding to the times in the resulting array. The masses of these nodes are linearly interpolated between the existing nodes on the branch. Once these new nodes have been added, all other nodes are removed from the tree[9] The processing is useful to construct representations of trees as they would be if only sparse time sampling were available. As such, it is useful for exploring how the number of snapshots in merger trees extracted from N-body simulations affects the properties of galaxies that form in them.

[9] The base node of the tree is never removed, even if it does not lie on one of the times in the constructed array.

12.38.3. Mass Accretion History Output

Output of the mass accretion history (i.e. the mass of the node on the primary branch as a function of time) for each merger tree can be requested by setting `[massAccretionHistoryOutput]=true`. If requested, an additional group, `massAccretionHistories`, is made in the GALACTICUS output file. This group will contain a subgroup for each merger tree (`mergerTreeN` where `N` is the merger tree index) within which three datasets, `nodeIndex`, `nodeTime` and `nodeMass`, can be found. These give the index, time and mass of the node on the primary branch of the tree at all times for which the tree is defined.

12.38.4. Tree Pruning By Mass

This task allows for branches of merger trees to be pruned—i.e. nodes below a specified mass limit are removed from the tree prior to any evolution. This can be useful for convergence studies for example. To prune branches set `[mergerTreePruneBranches]=true` and set `[mergerTreePruningMassThreshold]` to the desired mass threshold below which nodes will be pruned.

12.38.5. Tree Pruning By Hierarchy

This task allows for branches of merger trees to be pruned by hierarchy—i.e. nodes below a given depth in the hierarchy[10] are removed from the tree prior to any evolution. To prune branches by hierarchy depth set `[mergerTreePruneHierarchyAtDepth]` to the desired depth at which to prune. For example, a value of 1 will result in all branches except for the main branch being removed, while a value of 2 will remove all branches that do not merge directly onto the main branch. (Note that setting `[mergerTreePruneHierarchyAtDepth]` to zero will result in no pruning.)

12.38.6. Tree Pruning By Essential Node

This task allows for branches of merger trees to be pruned to leave only the branch occupied by an "essential" node, along with any progenitor branches of that node. This is intended primarily for debugging purposes where it can be convenient to cut away all non-essential branches of the tree. This pruning task is activated by setting `[mergerTreePruneNonEssential]=true`. The essential node is specified by `[mergerTreePruningNonEssentialID]` and `[mergerTreePruningNonEssentialTime]`, which specify the index of the node, and the time at which its properties are required. Specifying the time is important—if the node is a satellite at this time, then the pruning will not remove any progenitors of the parent node in which the essential node lives at the specified time.

12.39. Chemical Reaction Rates

Methods for computing chemical reaction rates are selected via the `[chemicalReactionRatesMethods]` parameter. Multiple methods can be selected—their rates are cumulated.

12.39.1. Null

Selected with `chemicalReactionRatesMethods=null` this is a null method which does not set any rates.

[10] The main branch is defined as depth 0. Other branches are assigned a depth equal to the depth of the branch onto which they merge plus 1. For example, any branch which merges directly onto the main branch is defined as depth 1.

12.39.2. Hydrogen Network

Selected with `chemicalReactionRatesMethods=hydrogenNetwork` this method computes rates using the network of reactions and fitting functions from Abel et al. [1997] and Tegmark et al. [1997]. The parameter `[hydrogenNetworkFast]` controls the approximations made. If set `true` then H^- is assumed to be at equilibrium abundance, H_2^+ reactions are ignored and other slow reactions are ignored (see Abel et al. 1997).

12.40. Ram Pressure Induced Mass Loss Rates in Disks/Spheroids

The methods for computing ram pressure induced rates of mass loss in disks/spheroids are selected via the `ramPressureStrippingMassLossRate(Disks|Spheroids)Method` parameter.

12.40.1. Simple

Selected with `ramPressureStrippingMassLossRate(Disks|Spheroids)Method=simple` this method computes the mass loss rate to be, for disks:

$$\dot{M}_{\rm gas,disk} = \min\left(\frac{\mathcal{F}_{\rm hot,host}}{2\pi {\rm G}\Sigma_{\rm gas}(r_{\rm half})\Sigma_{\rm total}(r_{\rm half})}, R_{\rm maximum}\right)\frac{M_{\rm gas,disk}}{\tau_{\rm dyn,disk}}, \tag{12.190}$$

where $\mathcal{F}_{\rm hot,host}$ is the ram pressure force due to the hot halo of the node's host (computed using the selected hot halo ram pressure force method; see §12.23), $\Sigma_{\rm gas}(r)$ is the gas surface density in the disk, $\Sigma_{\rm total}(r)$ is the total surface density in the disk, $r_{\rm half}$ is the disk half-mass radius, $M_{\rm gas,disk}$ is the total gas mass in the disk, $\tau_{\rm dyn,disk} = r_{\rm disk}/v_{\rm disk}$ is the dynamical time in the disk, and $R_{\rm maximum}$ =`[ramPressureStrippingMassLossRateDiskSimp` controls the maximum allowed rate of mass loss. For spheroids the mass loss rate is:

$$\dot{M}_{\rm gas} = -\max(\alpha, R_{\rm maximum})M_{\rm gas}/\tau_{\rm spheroid}, \tag{12.191}$$

where $R_{\rm maximum}$ =`[ramPressureStrippingMassLossRateSpheroidSimpleFractionalRateMax]`

$$\alpha = \mathcal{F}_{\rm hot,host}/F_{\rm gravity}, \tag{12.192}$$

and,

$$F_{\rm gravity} = \frac{4}{3}\rho_{\rm gas}(r_{1/2})\frac{{\rm G}M_{\rm total}(r_{1/2})}{r_{1/2}} \tag{12.193}$$

is the gravitational restoring force in the spheroid at the half-mass radius, $r_{1/2}$ [Takeda et al., 1984].

12.40.2. Null

Selected with `ramPressureStrippingMassLossRate(Disks|Spheroids)Method=null` this method assumes zero mass loss rate always.

12.41. Satellite Dynamical Friction

The method to be used for computing the satellite vector acceleration due to dynamical friction is specified by the `satelliteDynamicalFrictionMethod` parameter.

12.41.1. Null

Selected with `satelliteDynamicalFrictionMethod=null`, this method assumes that there is no dynamical friction in the system.

12.41.2. Chandrasekhar (1943)

Selected with `satelliteDynamicalFrictionMethod=Chandrasekhar1943`, this method uses the Chandrasekhar [1943] formula to compute the acceleration of a satellite at radius r from the center of the host due to dynamical friction:

$$\mathbf{a}_{DF} = -\frac{4\pi \mathrm{G}^2 M_{\rm sat} \rho_{\rm host}(r)}{v_{\rm sat}^3} \ln\Lambda \left[\mathrm{erf}(x) - \frac{2x}{\sqrt{\pi}} \exp(-x^2)\right] \mathbf{v}_{\rm sat}, \tag{12.194}$$

where $M_{\rm sat}$ and $\mathbf{v}_{\rm sat}$ are the satellite's mass and velocity, respectively, $v_{\rm sat} = |\mathbf{v}_{\rm sat}|$, $\rho_{\rm host}(r)$ is the host's density profile, $\ln\Lambda =$`[satelliteDynamicalFrictionChandrasekharCoulombLogarithm]` is the Coulomb logarithm, and $x \equiv v_{\rm sat}/\sqrt{2}\sigma(r)$, where $\sigma(r)$ is the velocity dispersion of the host halo at radius r, approximated to be equal to the host virial velocity, $v_{\rm vir}$.

12.42. Satellite Tidal Fields

The method for computing the tidal field experienced by satellite galaxies is selected via `satellitesTidalFieldMethod`.

12.42.1. Null

Selected with `satellitesTidalFieldMethod=null` this method assumes a zero tidal field always.

12.42.2. Spherical Symmetry

Selected with `satellitesTidalFieldMethod=sphericalSymmetry` this method assumes a spherically-symmetric host halo, and computes the tidal field accordingly using:

$$\mathcal{F} = \frac{\mathrm{G}M_{\rm host}(< r_{\rm p})}{r_{\rm p}^3} - 4\pi \mathrm{G}\rho_{\rm host}(r_{\rm p}) + \omega_{\rm p}^2, \tag{12.195}$$

where $r_{\rm p}$ is the pericentric radius. $M_{\rm host}(< r)$ is the mass of the host halo enclosed within a sphere of radius r, $\rho_{\rm host}(r)$ is the host density at radius r, and $\omega_{\rm p}$ is the orbital angular velocity at pericenter.

12.43. Satellite Tidal Stripping

The method to be used for computing the satellite mass loss rate due to tidal stripping is specified by the `satelliteTidalStrippingMethod` parameter.

12.43.1. Null

Selected with `satelliteTidalStrippingMethod=null`, this method assumes that there is no tidal stripping in the system.

12.43.2. Zentner (2005)

Selected by default with `satelliteTidalStrippingMethod=Zentner2005`, this method uses the formalism of Zentner et al. [2005] to compute the mass loss rate $\dot{M}_{\rm sat}$:

$$\dot{M}_{\rm sat} = -\alpha \frac{M_{\rm sat}(> r_{\rm tidal})}{T_{\rm orb}}, \tag{12.196}$$

where $\alpha =$ `[satelliteTidalStrippingZentnerRate]`,

$$T_{\rm orb} = \frac{1}{\max(\omega/2\pi, v_{\rm r}/r)}, \tag{12.197}$$

where ω is the angular velocity of the satellite, $v_{\rm r}$ is the radial velocity, r is the orbital radius, and $r_{\rm tidal}$ is the tidal radius of the satellite, given by the King [1962] formula:

$$r_{\rm tidal} = \left(\frac{GM_{\rm sat}}{\omega^2 - d^2\Phi/dr^2} \right)^{1/3}, \tag{12.198}$$

where ω is the orbital angular velocity of the satellite, and $\Phi(r)$ is the gravitational potential due to the host.

12.44. Satellite Tidal Heating

The method to be used for computing the integrated, normalized (i.e. the energy divided by radius squared) tidal heating energy, $Q_{\rm tidal}$, for subhalos is specified by the `satelliteTidalHeatingMethod` parameter.

12.44.1. Null

Selected with `satelliteTidalHeatingMethod=null`, this method assumes that there is no tidal heating.

12.44.2. Gnedin (1999)

Selected by default with `satelliteTidalHeatingMethod=Gnedin1999`, this method uses the formalism of Gnedin et al. [1999] to compute the heating rate:

$$\dot{Q}_{\rm tidal} = \frac{1}{3}\epsilon \left[1 + \left(\frac{T_{\rm shock}}{T_{\rm orb}} \right)^2 \right]^{-\gamma} g_{ij} G^{ij} \tag{12.199}$$

where $T_{\rm orb}$ and $T_{\rm shock}$ are the orbital period and shock duration, respectively, of the satellite, $\epsilon =$ `[satelliteTidalHeatingGne` and $\gamma =$ `[satelliteTidalHeatingGnedinGamma]` are model parameters, g_{ij} is the tidal tensor, and G_{ij} is the integral with respect to time of g_{ij} along the orbit of the satellite. Upon tidal heating, a mass element at radius $r_{\rm i}$ expands to radius $r_{\rm f}$, according to the equation

$$\frac{1}{r_{\rm f}} = \frac{1}{r_{\rm i}} - \frac{2r_{\rm i}^3 Q_{\rm tidal}}{{\rm G}M_{\rm sat}(< r_{\rm i})}. \tag{12.200}$$

12.45. Star Formation Rate Surface Densities

The method for computing surface densities of star formation rate in disks is selected via `starFormationRateSurfaceDensityI`

12.45.1. Kennicutt-Schmidt

Selected with `starFormationRateSurfaceDensityDisksMethod=Kennicutt-Schmidt` this method assumes that the Kennicutt-Schmidt law holds [Schmidt, 1959, Kennicutt, 1998]:

$$\dot{\Sigma}_\star = A \left(\frac{\Sigma_{\rm H}}{M_\odot {\rm pc}^{-2}} \right)^N, \tag{12.201}$$

where $A =$ `[starFormationKennicuttSchmidtNormalization]` and $N =$ `[starFormationKennicuttSchmidtExponent]` are parameters. Optionally, if the `[starFormationKennicuttSchmidtTruncate]` parameter is set to true, then the star formation rate is truncated below a critical surface density such that

$$\dot{\Sigma}_\star = \begin{cases} A \left(\frac{\Sigma_{\rm H}}{M_\odot {\rm pc}^{-2}} \right)^N & \text{if } \Sigma_{\rm gas,disk} > \Sigma_{\rm crit} \\ A \left(\frac{\Sigma_{\rm H}}{M_\odot {\rm pc}^{-2}} \right)^N (\Sigma_{\rm gas,disk}/\Sigma_{\rm crit})^\alpha & \text{otherwise.} \end{cases} \tag{12.202}$$

Here, $\alpha =$ `[starFormationKennicuttSchmidtExponentTruncated]` and $\Sigma_{\rm crit}$ is a critical surface density for star formation which we specify as

$$\Sigma_{\rm crit} = \frac{q_{\rm crit} \kappa \sigma_{\rm gas}}{\pi {\rm G}}, \tag{12.203}$$

where κ is the epicyclic frequency in the disk, $\sigma_{\rm gas}$ is the velocity dispersion of gas in the disk and $q_{\rm crit} =$ `[toomreParameterCritical]` is a dimensionless constant of order unity which controls where the critical density occurs. We assume that $\sigma_{\rm gas}$ is a constant equal to `[velocityDispersionDiskGas]` and that the disk has a flat rotation curve such that $\kappa = \sqrt{2} V/R$.

12.45.2. Extended Schmidt

Selected with `starFormationRateSurfaceDensityDisksMethod=extendedSchmidt` this method assumes that the extended Schmidt law holds [Shi et al., 2011]:

$$\dot{\Sigma}_\star = A \left(x_{\rm H} \frac{\Sigma_{\rm gas}}{M_\odot {\rm pc}^{-2}} \right)^{N_1} \left(\frac{\Sigma_\star}{M_\odot {\rm pc}^{-2}} \right)^{N_2} \tag{12.204}$$

where $A =$ `[starFormationExtendedSchmidtNormalization]`, $N_1 =$ `[starFormationExtendedSchmidtGasExponent]` and $N_2 =$ `[starFormationExtendedSchmidtStarExponent]` are parameters.

12.45.3. Blitz-Rosolowsky

Selected with `starFormationRateSurfaceDensityDisksMethod=Blitz-Rosolowsky2006` this method assumes that the star formation rate is given by [Blitz and Rosolowsky, 2006]:

$$\dot{\Sigma}_\star(R) = \nu_{\rm SF}(R) \Sigma_{\rm H_2,disk}(R), \tag{12.205}$$

where $\nu_{\rm SF}$ is a frequency given by

$$\nu_{\rm SF}(R) = \nu_{\rm SF,0} \left[1 + \left(\frac{\Sigma_{\rm HI}}{\Sigma_0} \right)^q \right], \tag{12.206}$$

where $q =$ `[surfaceDensityExponentBlitzRosolowsky]` and $\Sigma_0 =$ `[surfaceDensityCriticalBlitzRosolowsky]` are parameters and the surface density of molecular gas $\Sigma_{\rm H_2} = (P_{\rm ext}/P_0)^\alpha \Sigma_{\rm HI}$, where $\alpha =$ `[pressureExponentBlitzRosolowsk` and $P_0 =$ `[pressureCharacteristicBlitzRosolowsky]` are parameters and the hydrostatic pressure in the disk plane assuming location isothermal gas and stellar components is given by

$$P_{\rm ext} \approx \frac{\pi}{2} {\rm G} \Sigma_{\rm gas} \left[\Sigma_{\rm gas} + \left(\frac{\sigma_{\rm gas}}{\sigma_\star} \right) \Sigma_\star \right] \tag{12.207}$$

where we assume that the velocity dispersion in the gas is fixed at $\sigma_{\rm gas} =$ `[velocityDispersionDiskGas]` and, assuming $\Sigma_\star \gg \Sigma_{\rm gas}$, we can write the stellar velocity dispersion in terms of the disk scale height, $h_\star$, as

$$\sigma_\star = \sqrt{\pi {\rm G} h_\star \Sigma_\star} \tag{12.208}$$

where we assume $h_\star / R_{\rm disk} =$ `[heightToRadialScaleDiskBlitzRosolowsky]`.

12.45.4. Krumholz-McKee-Tumlinson

Selected with `starFormationRateSurfaceDensityDisksMethod=KMT09` this method assumes that the star formation rate is given by [Krumholz et al., 2009]:

$$\dot{\Sigma}_\star(R) = \nu_{\rm SF} f_{\rm H_2}(R) \Sigma_{\rm HI,disk}(R) \left\{ \begin{array}{ll} (\Sigma_{\rm HI}/\Sigma_0)^{-1/3}, & \mathrm{if}\ \Sigma_{\rm HI}/\Sigma_0 \leq 1 \\ (\Sigma_{\rm HI}/\Sigma_0)^{1/3}, & \mathrm{if}\ \Sigma_{\rm HI}/\Sigma_0 > 1 \end{array} \right. , \tag{12.209}$$

where $\nu_{\rm SF}$ =`[starFormationFrequencyKMT09]` is a frequency and $\Sigma_0 = 85 M_\odot {\rm pc}^{-2}$. The molecular fraction is given by

$$f_{\rm H_2} = 1 - \left(1 + \left[\frac{3s}{4(1+\delta)} \right]^{-5} \right)^{-1/5} , \tag{12.210}$$

where

$$\delta = 0.0712 \left[0.1 s^{-1} + 0.675 \right]^{-2.8} , \tag{12.211}$$

and

$$s = \frac{\ln(1 + 0.6\chi + 0.01\chi^2)}{0.04 \Sigma_{\rm comp,0} Z'}, \tag{12.212}$$

with

$$\chi = 0.77 \left[1 + 3.1 Z'^{0.365} \right] , \tag{12.213}$$

and $\Sigma_{\rm comp,0} = c \Sigma_{\rm HI} / M_\odot {\rm pc}^{-2}$ where c =`[molecularComplexClumpingFactorKMT09]` is a density enhancement factor relating the surface density of molecular complexes to the gas density on larger scales. Alternatively, if `[molecularFractionFastKMT09]` is set to true, the molecular fraction will be computed using the faster (but less acccurate at low molecular fraction) formula

$$f_{\rm H_2} = 1 - \frac{3s/4}{(1 + s/4)}. \tag{12.214}$$

12.46. Star Formation Timescales

The methods for computing star formation timescales in disks and spheroids are selected via the `starFormationTimescaleDisk` and `starFormationTimescaleSpheroidsMethod` respectively.

12.46.1. Fixed

Selected with `starFormationTimescaleDisksMethod=fixed` this method assumes a fixed timescale for star formation `[starFormationTimescaleDisksFixedTimescale]` (in Gyr).

12.46.2. Halo Scaling

Selected with `starFormationTimescaleDisksMethod=haloScaling` this method assumes a timescale for star formation of

$$\tau_\star = \tau_{\star,0} \left(\frac{V_{\rm vir}}{200 {\rm km/s}} \right)^{\alpha_\star} (1+z)^{\beta_\star}, \tag{12.215}$$

where $\tau_{\star,0}$ =`[starFormationTimescaleDisksHaloScalingTimescale]`, $\alpha_\star$ =`[starFormationTimescaleDisksHaloScaling` and $\beta_\star$ =`[starFormationTimescaleDisksHaloScalingRedshiftExponent]`.

12.46.3. Dynamical Time

Selected with `starFormationTimescale[Disks|Spheroids]Method=dynamicalTime` this method computes the star formation timescale to be:

$$\tau_\star = \epsilon_\star^{-1} \tau_{\rm dynamical} \left(\frac{V}{200{\rm km/s}} \right)^{\alpha_\star}, \tag{12.216}$$

where $\epsilon_\star =$`starFormation[Disks|Spheroids]Efficiency` and $\alpha_\star =$`starFormation[Disks|Spheroids]VelocityExponent` are input parameters, $\tau_{\rm dynamical} \equiv r/V$ is the dynamical timescale of the component and r and V are the characteristic radius and velocity respectively of the component. The timescale is not allowed to fall below a minimum value specified by `starFormation[Disks|Spheroids]MinimumTimescale` (in Gyr).

12.46.4. Integrated Surface Density

Selected with `starFormationTimescaleDisksMethod=integratedSurfaceDensity` this method computes the star formation timescale to be:

$$\tau_\star = \frac{M_{\rm cold}}{\int_0^\infty 2\pi r \dot{\Sigma}_\star(r) {\rm d}r}, \tag{12.217}$$

where $\dot{\Sigma}_\star(r)$ is the surface density of star formation rate (see §12.45).

12.46.5. Baugh et al. (2005)

Selected with `starFormationTimescaleDisksMethod=Baugh2005` this method assumes that the star formation rate is given by a modified version of the Baugh et al. [2005] prescription:

$$\tau_\star = \tau_0 (V_{\rm disk}/200{\rm km/s})^\alpha a^\beta \tag{12.218}$$

where $\tau_0 =$`[starFormationDiskTimescale]`, $\alpha =$`[starFormationDiskVelocityExponent]` and $\beta =$`[starFormationExpans`

12.47. Stellar Population Properties

Algorithms for determining stellar population properties—essentially the rates of change of stellar and gas mass and abundances given a star formation rate and fuel abundances (and perhaps a historical record of star formation in the component)—are selected by the `stellarPopulationPropertiesMethod` parameter.

12.47.1. Instantaneous

Selected with `stellarPopulationPropertiesMethod=instantaneous` this method uses the instantaneous recycling approximation. Specifically, given a star formation rate ϕ, this method assumes a rate of increase of stellar mass of $\dot{M}_\star = (1-R)\phi$, a corresponding rate of decrease in fuel mass. The rate of change of the metal content of stars follows from the fuel metallicity, while that of the fuel changes according to

$$\dot{M}_{fuel,Z} = -(1-R) Z_{\rm fuel} \phi + p\phi. \tag{12.219}$$

In the above R is the instantaneous recycled fraction and p is the yield, both of which are supplied by the IMF subsystem. The rate of energy input from the stellar population is computed assuming that the canonical amount of energy from a single stellar population (as defined by the `feedbackEnergyInputAtInfinityCanonical`) is input instantaneously.

12.47.2. Noninstantaneous

Selected with `stellarPopulationPropertiesMethod=noninstantaneous` this method assumes fully non-instantaneous recycling and metal enrichment. Recycling and metal production rates from simple stellar populations are computed, for any given IMF, from stellar evolution models. The rates of change are then:

$$\dot{M}_\star = \phi - \int_0^t \phi(t')\dot{R}(t-t';Z_{\rm fuel}[t'])\mathrm{d}t', \tag{12.220}$$

$$\dot{M}_{\rm fuel} = -\phi + \int_0^t \phi(t')\dot{R}(t-t';Z_{\rm fuel}[t])\mathrm{d}t', \tag{12.221}$$

$$\dot{M}_{\star,Z} = Z_{\rm fuel}\phi - \int_0^t \phi(t')Z_{\rm fuel}(t')\dot{R}(t-t';Z_{\rm fuel}[t'])\mathrm{d}t', \tag{12.222}$$

$$\dot{M}_{{\rm fuel},Z} = -Z_{\rm fuel}\phi + \int_0^t \phi(t')\{Z_{\rm fuel}(t')\dot{R}(t-t';Z_{\rm fuel}[t']) + \dot{p}(t-t';Z_{\rm fuel}[t'])\}\mathrm{d}t', \tag{12.223}$$

$$\tag{12.224}$$

where $\dot{R}(t;Z)$ and $\dot{p}(t;Z)$ are the recycling and metal yield rates respectively from a stellar population of age t and metallicity Z. The energy input rate is computed self-consistently from the star formation history.

12.48. Stellar Population Spectra

Stellar population spectra are used to construct intgrated spectra of galaxies. The method used to compute such spectra is specified by the `stellarPopulationSpectraMethod` parameter.

12.48.1. Conroy, White & Gunn (2009)

Selected with `stellarPopulationSpectraMethod=Conroy-White-Gunn2009` this method uses v2.2 of the FSPS code of Conroy et al. [2009] to compute stellar spectra. If necessary, the FSPS code will be downloaded, patched and compiled and run to generate spectra. These tabulations are then stored to file for later retrieval. The file name used is `datasets/dynamic/stellarPopulations/simpleStellarPopulationsFSPS:v2.5_-<descriptor>.hdf5` where `<descriptor>` is an MD5 hash descriptor of the selected stellar population.

12.48.2. File

Selected with `stellarPopulationSpectraMethod=file` this method reads stellar population spectra from an HDF5 file, with format described in §16.4.1.

12.49. Stellar Population Spectra Postprocessing

Stellar population spectra are postprocessed (to handle, for example, absorption by the IGM).

Different chains of postprocessing methods can be applied to different filters. The `[luminosityPostprocessSet]` parameter specifies, for each filter, which chain of postprocessing filters to use. (If this parameter is not present then "`default`" is assumed for all filters.) The filters used in each chain are specified by the input parameter `[stellarPopulationSpectraPostprocess<setName>Method]` where `<setName>` is the name specified in `[luminosityPostprocessSet]`.

12.49.1. Inoue, Shimizu & Iwata (2014) IGM Attenuation

Selected with `stellarPopulationSpectraPostprocess<setName>Method=inoue2014` this method post-processes spectra through absorption by the IGM using the results of Inoue et al. [2014].

12.49.2. Meiksin (2006) IGM Attenuation

Selected with `stellarPopulationSpectraPostprocess<setName>Method=meiksin2006` this method post-processes spectra through absorption by the IGM using the results of Meiksin [2006].

12.49.3. Madau (1995) IGM Attenuation

Selected with `stellarPopulationSpectraPostprocess<setName>Method=madau1995` this method post-processes spectra through absorption by the IGM using the results of Madau [1995].

12.49.4. Lyman-continuum Suppression

Selected with `stellarPopulationSpectraPostprocess<setName>Method=lycSuppress` this method suppresses all emission in the Lyman continuum.

12.49.5. Recent Star Formation

Selected with `stellarPopulationSpectraPostprocess<setName>Method=recent` this method suppresses all emission from populations older than `[recentPopulationsTimeLimit]`.

12.49.6. Identity

Selected with `stellarPopulationSpectraPostprocess<setName>Method=identity` this method leaves the spectrum unchanged.

12.50. Stellar Astrophysics

Various properties related to stellar astrophysics are required by Galacticus. The following documents their implementation.

12.50.1. Basics

This subset of properties include recycled mass, metal yield and lifetime. The method used to compute such properties is specified by the `stellarAstrophysicsMethod` parameter.

File

Selected with `stellarAstrophysicsMethod=file` this method uses reads properties of individual stars of different initial mass and metallicity from an XML file and interpolates in them. The stars can be irregularly spaced in the plane of initial mass and metallicity. The XML file should have the following structure:

```
<stars>
 <star>
   <initialMass>0.6</initialMass>
   <lifetime>28.19</lifetime>
```

```
    <metallicity>0.0000</metallicity>
    <ejectedMass>7.65</ejectedMass>
    <metalYieldMass>0.44435954</metalYieldMass>
    <elementYieldMassFe>2.2017e-13</elementYieldMassFe>
    <source>Table 2 of Tumlinson, Shull &amp;amp; Venkatesan (2003, ApJ, 584, 608)</source>
    <url>http://adsabs.harvard.edu/abs/2003ApJ...584..608T</url>
  </star>
  <star>
    .
    .
    .
  </star>
  .
  .
  .
 </stars
```

Each `star` element must contain the `initialMass` (given in $M_\odot$) and `metallicity` tags. Other tags are optional. `lifetime` gives the lifetime of such a star (in Gyr), `ejectedMass` gives the total mass (in $M_\odot$) ejected by such a star during its lifetime, `metalYieldMass` gives the total mass of metals yielded by the star during its lifetime while `elementYieldMassX` gives the mass of element `X` yielded by the star during its lifetime. The `source` and `url` tags are not used, but are strongly recommended to provide a reference to the origin of the stellar data.

12.50.2. Stellar Winds

Energy input to the ISM from stellar winds is used in calculations of feedback efficiency. The method used to compute stellar wind properties is specified by the `stellarWindsMethod` parameter.

Leitherer et al. (1992)

Selected with `stellarWindsMethod=Leitherer1992` this method uses the fitting formulae of Leitherer et al. [1992] to compute stellar wind energy input from the luminosity and effective temperature of a star.

12.50.3. Stellar Tracks

The method used to compute stellar tracks is specified by the `stellarTracksMethod` parameter.

File

Selected with `stellarTracksMethod=file` in this method luminosities and effective temperatures of stars are computed from a tabulated set of stellar tracks. The file containing the tracks to use is specified via the `stellarTracksFile` parameter. The file specified must be an HDF5 file with the following structure:

```
stellarTracksFile
 |
 +-> metallicity1
 |    |
 |    +-> metallicity
 |    |
```

```
|    +-> mass1
|    |    |
|    |    +-> mass
|    |    |
|    |    +-> age
|    |    |
|    |    +-> luminosity
|    |    |
|    |    +-> effectiveTemperature
|    |
|    x-> massN
|
x-> metallicityN
```

Each `metallicityN` group tabulates tracks for a given metallicity (the value of which is stored in the `metallicity` dataset within each group), and may contain an arbitrary number of `massN` groups. Each `massN` group should contain a track for a star of some mass (the value of which is given in the `mass` dataset). Within each track three datasets specify the `age` (in Gyr), `luminosity` (in $L_\odot$) and `effectiveTemperature` (in Kelvin) along the track.

12.50.4. Supernovae Type Ia

Properties of Type Ia supernovae, including the cumulative number occuring and metal yield, are handled by the method selected using the `supernovaeIaMethod` parameter.

Nagashima et al. (2005) Prescription

Selected with `supernovaeIaMethod`=`Nagashima` this method uses the prescriptions from Nagashima et al. [2005] to compute the numbers and yields of Type Ia supernovae.

12.50.5. Population III Supernovae

Properties of Population III specific supernovae are handled by the method selected with the `supernovaePopIIIMethod` parameter.

Heger & Woosley (2002)

Selected with `supernovaePopIIIMethod`=`Heger-Woosley2002` this method computes the energies of pair instability supernovae from the results of Heger and Woosley [2002].

12.50.6. Stellar Feedback

Aspects of stellar feedback are computed by the method selected with the `stellarFeedbackMethod` parameter.

Standard

Selected with `stellarFeedbackMethod`=`standard`, the method assumes that the cumulative energy input from a stellar population is equal to the total number of (Type II and Type Ia) supernovae multiplied by `supernovaEnergy` (specified in ergs) plus any Population III-specific supernovae energy plus the integrated energy input from stellar winds. The minimum mass of a star required to form a Type II supernova is specified (in $M_\odot$) via the `initialMassForSupernovaeTypeII` parameter.

12.51. Substructure and Merging

Substructures and merging of nodes/substructures is controlled by several algorithms which are described below:

12.51.1. Merging Timescales

The method used to compute merging timescales of substructures is specified by the `satelliteMergingTimescalesMethod` parameter.

Lacey & Cole (1993)

Selected with `satelliteMergingTimescalesMethod=laceyCole1993`, this method computes merging timescales using the dynamical friction calculation of Lacey and Cole [1993]. Timescales are multiplied by the value of the `mergingTimescaleMultiplier` input parameter.

Lacey & Cole (1993) + Tormen

Selected with `satelliteMergingTimescalesMethod=laceyCole1993Tormen`, this method computes merging timescales using the dynamical friction calculation of Lacey and Cole [1993] with a parameterization of orbital parameters designed to fit the results of Tormen [1997] as described by Cole et al. [2000]. Timescales are multiplied by the value of the `mergingTimescaleMultiplier` input parameter. Specifically, the merging time is taken to be:

$$\tau_\mathrm{merge} = \frac{f_\tau \Phi \tau_\mathrm{dynamical}}{2B(1)} \frac{M_\mathrm{host}/M_\mathrm{satellite}}{\ln(M_\mathrm{host}/M_\mathrm{satellite})} \tag{12.225}$$

where $f_\tau =$`mergingTimescaleMultiplier`, $\tau_\mathrm{dynamical}$ is the dynamical time of the host halo and $B(x) = \mathrm{erf}(x) - 2x\exp(x)/\sqrt{\Pi}$. The orbital factor $\Phi \equiv \epsilon^{0.78}(R_\mathrm{c}/R_\mathrm{virial})^2$ is drawn at random from a log-normal distribution with median -0.14 and dispersion 0.26 as found by Cole et al. [2000].

Jiang (2008)

Selected with `satelliteMergingTimescalesMethod=jiang2008`, this method computes merging timescales using the dynamical friction calibration of Jiang et al. [2008]. Jiang et al. [2008] find that their fitting formula does not perfectly capture the results of N-body simulations, instead showing a scatter in the ratio $T_\mathrm{fit}/T_\mathrm{sim}$ where T_fit is the merging time from their fitting formula and T_sim is that measured from their N-body simulation. Furthermore, they show that the distribution of $T_\mathrm{fit}/T_\mathrm{sim}$ is well described by a log-normal with dispersion (in $\log[T_\mathrm{fit}/T_\mathrm{sim}]$) of $\sigma = 0.4$. Random perturbations can be applied to the merger times returned by this implementation by setting $\sigma =$`[satelliteMergingJiang2008Scatter]` > 0 which will cause the merger time to be drawn from a log-normal distribution of width σ with median equal to T_fit.

Boylan-Kolchin (2008)

Selected with `satelliteMergingTimescalesMethod=boylanKolchin2008`, this method computes merging timescales using the dynamical friction calibration of Boylan-Kolchin et al. [2008].

Wetzel & White (2010)

Selected with `satelliteMergingTimescalesMethod=wetzelWhite2010`, this method computes merging timescales using the dynamical friction calibration of Wetzel and White [2010].

Villalobos et al. (2013)

Selected with `satelliteMergingTimescalesMethod=villalobos2013`, this method computes merging timescales using the modifier of Villalobos et al. [2013] as

$$\tau_{\rm merge} = (1+z)^{\alpha}\tau'_{\rm merge}, \tag{12.226}$$

where $\alpha =$ `[satelliteMergingTimescaleVillalobos2013Exponent]` and $\tau'_{\rm merge}$ is the merging timescale computed by another satellite merging timescale method as specified via `[satelliteMergingTimescaleVillalobos2013Base`

Preset

Selected with `satelliteMergingTimescalesMethod=preset`, this method returns a precomputed merging timescale stored by the satellite component.

Null

Selected with `satelliteMergingTimescalesMethod=null`, this method returns zero merging timescale.

Infinite

Selected with `satelliteMergingTimescalesMethod=infinite`, this method returns an infinite merging timescale (technically, it returns 10^{30} Gyr which should be sufficiently close to infinity for practical purposes).

12.51.2. Virial Orbits

The algorithm to be used to determine orbital parameters of substructures when they first enter the virial radius of their host is specified via the `virialOrbitMethod` parameter.

Benson (2005)

Selected with `virialOrbitMethod=benson2005`, this method selects orbital parameters randomly from the distribution given by Benson [2005]. If the virial density contrast definition differs from that used by Benson [2005] then the orbit is assigned based on Benson [2005]'s definition and then propagated to the virial radius relevant to the current definition of density contrast.

Fixed

Selected with `virialOrbitMethod=fixed`, this method sets all orbital parameters to fixed values, with $v_{\rm r} =$ `[virialOrbitsFixedRadialVelocity]` $V_{\rm virial}$ and $v_\phi =$ `[virialOrbitsFixedTangentialVelocity]` $V_{\rm virial}$.

Wetzel (2010)

Selected with `virialOrbitMethod=wetzel2010`, this method selects orbital parameters randomly from the distribution given by Wetzel [2010], including the redshift and mass dependence of the distributions. Note that the parameter R_1 can become negative (which is unphysical) for certain regimes of mass and redshift according to the fitting function for R_1 given by Wetzel [2010]. Therefore, we enforce $R_1 > 0.05$. Similarly, the parameter C_1 can become very large in some regimes which is probably an artifact of the fitting function used rather than physically meaningful (and which causes numerical difficulties in evaluating the distribution). We therefore prevent C_1 from exceeding 9.999999[11] If the virial density

[11] We use this value rather than 10 since the GSL ${}_2F_1$ hypergeomtric function fails in some cases when $C_1 \geq 10$.

contrast definition differs from that used by Wetzel [2010] then the orbit is assigned based on Wetzel [2010]'s definition and then propagated to the virial radius relevant to the current definition of density contrast.

Jiang et al. (2014)

Selected with `virialOrbitMethod=jiang2014`, this method selects orbital parameters randomly from the distribution given by Jiang et al. [2014], including the mass and mass-ratio dependence of the distributions. If the virial density contrast definition differs from that used by Jiang et al. [2014] then the orbit is assigned based on Jiang et al. [2014]'s definition and then propagated to the virial radius relevant to the current definition of density contrast.

12.51.3. Node Merging

The algorithm to be used to process nodes when they become substructures is specified by the `nodeMergersMethod` parameter.

Single Level Hierarchy

Selcted with `nodeMergersMethod=singleLevelHierarchy`, this method maintains a single level hierarchy of substructure, i.e. it tracks only substructures, not sub-substructures or deeper levels. When a node first becomes a satellite it is appended to the list of satellites associated with its host halo. If the node contains its own satellites they will be detached from the node and appended to the list of satellites of the new host (and assigned new merging times).

12.52. Supernovae Feedback Models

The supernovae feedback driven outflow rate is computed using the method specified by the `starFormationFeedback[Disks|S` for disks and spheroids respectively.

12.52.1. Fixed

Selected with `starFormationFeedbackDisksMethod=fixed`, this method assumes an outflow rate of:

$$\dot{M}_{\rm outflow} = f_{\rm outflow} \frac{\dot{E}}{E_{\rm canonical}}, \tag{12.227}$$

where $f_{\rm outflow}$ =`[diskOutflowFraction]` is the fraction of the star formation rate that goes into outflow, $\dot{E}$ is the rate of energy input from stellar populations and $E_{\rm canonical}$ is the total energy input by a canonical stellar population normalized to $1 M_\odot$ after infinite time.

12.52.2. Power Law

Selected with `starFormationFeedback[Disks|Spheroids]Method=powerLaw`, this method assumes an outflow rate of:

$$\dot{M}_{\rm outflow} = \left(\frac{V_{\rm outflow}}{V} \right)^{\alpha_{\rm outflow}} \frac{\dot{E}}{E_{\rm canonical}}, \tag{12.228}$$

where $V_{\rm outflow}$ =`[disk|spheroid]OutflowVelocity` (in km/s) and $\alpha_{\rm outflow}$ =`[disk|spheroid]OutflowVelocity` are input parameters, V is the characteristic velocity of the component, $\dot{E}$ is the rate of energy input from stellar populations and $E_{\rm canonical}$ is the total energy input by a canonical stellar population normalized to $1 M_\odot$ after infinite time.

12.52.3. Creasey et al. (2012)

Selected with `starFormationFeedbackDisksMethod=Creasey2012`, this method computes the outflow rate using the model of Creasey et al. [2012]. Specifically,

$$\dot{M}_{\rm outflow} = \frac{\dot{E}_{\rm SN}}{E_{\rm SN}\dot{M}_\star}\int_0^\infty \beta_0 \Sigma_{g,1}^{-\mu}(r) f_{\rm g}^{\nu}(r)\dot{\Sigma}_\star(r) 2\pi r {\rm d}r, \tag{12.229}$$

where $\Sigma_{g,1}(r)$ is the surface density of gas in units of $M_\odot$ pc^{-2}, $f_{\rm g}(r)$ is the gas fraction, $\dot{\Sigma}_\star(r)$ is the surface density of star formation rate, $\dot{M}_\star$ is the total star formation rate in the disk, $\dot{E}_{\rm SN}$ is the current energy input rate from supernovae, $E_{\rm SN}$ is the total energy input per unit mass from a stellar population after infinite time, $\beta_0 =$`[starFormationFeedbackDisksCreasy2012Beta0]`, $\mu =$`[starFormationFeedbackDisksCreasy2012Mu]`, and $\nu =$`[starFormationFeedbackDisksCreasy2012Nu]`.

12.53. Supernovae Expulsive Feedback Models

The expulsive supernovae feedback driven outflow rate is computed using the method specified by the `starFormationExpulsiveFeedback[Disks|Spheroids]Method` for disks and spheroids respectively.

12.53.1. Null

Selected with `starFormationExpulsiveFeedback[Disks|Spheroids]Method=null`, this method assumes a zero outflow rate.

12.53.2. Superwind

Selected with `starFormationExpulsiveFeedback[Disks|Spheroids]Method=superwind`, this method assumes an outflow rate of:

$$\dot{M}_{\rm outflow} = \beta_{\rm superwind}\frac{\dot{E}}{E_{\rm canonical}}\begin{cases}(V_{\rm superwind}/V)^2 & \text{if } V > V_{\rm superwind} \\ 1 & \text{otherwise,}\end{cases} \tag{12.230}$$

where $V_{\rm superwind} =$`[disk|spheroid]SuperwindVelocity` (in km/s) and $\beta_{\rm superwind} =$`[disk|spheroid]SuperwindMassLoadin` are input parameters, V is the characteristic velocity of the component, $\dot{E}$ is the rate of energy input from stellar populations and $E_{\rm canonical}$ is the total energy input by a canonical stellar population normalized to $1M_\odot$ after infinite time.

12.54. Supermassive Black Hole Binaries: Initial Separation

The method to be used for computing the initial separation of black hole binaries is specified by the `blackHoleBinaryInitialRadiiMethod` parameter.

12.54.1. Spheroid Radius Fraction

Selected with `blackHoleBinaryInitialRadiiMethod=spheroidRadiusFraction`, this method assumes that the initial separation of the binary is equal to a fixed fraction `[blackHoleInitialRadiusSpheroidRadiusRatio]` of the larger of the spheroid scale radii of the two merging galaxies.

12.54.2. Volonteri (2003)

Selected with `blackHoleBinaryInitialRadiiMethod=Volonteri2003`, this method assumes that the initial separation follows the relationship described in Volonteri et al. [2003]

$$r_{\rm initial} = \frac{\rm G(M_{\bullet,1} + M_{\bullet,2})}{2\sigma^2_{\rm DM}} \tag{12.231}$$

where $M_{\bullet,1}$ and $M_{\bullet,2}$ are the masses of the black holes and $\sigma_{\rm DM}$ is the velocity dispersion of the dark matter, which we assume to equal the virial velocity of the dark matter halo.

12.54.3. Tidal Radius

Selected with `blackHoleBinaryInitialRadiiMethod=tidalRadius`, this method assumes an initial separation that corresponds to the distance at which the satellite galaxy is tidally stripped to its half-mass radius, thus only leaving the central massive black hole. Specifically, the initial radius is given by:

$$\frac{M_{\rm sat}}{2r^3_{\rm sat,1/2}} = -\frac{\rm d}{{\rm d}r}\frac{M_{\rm host}(r_{\rm initial})}{r^2_{\rm initial}} \tag{12.232}$$

Where $M_{\rm sat}$ is the mass of the satellite galaxy, $r_{\rm sat,1/2}$ is its half mass radius, $M_{\rm host}(r)$ is the mass of the host galaxy within radius r and $r_{\rm initial}$ is the initial radius.

12.55. Supermassive Black Hole Binaries: Separation Growth Rate

The method to be used for computing the separation growth rate of black hole binaries is specified by the `blackHoleBinarySeparationGrowthRateMethod` parameter.

12.55.1. Null

Selected by default and with `blackHoleBinarySeparationGrowthRateMethod=null`, this method assumes that the initial separation of the binaries is final.

12.55.2. Standard

Selected with `blackHoleBinarySeparationGrowthRateMethod=Standard`, this method computes the separation growth rate of the binaries following a modified version of Volonteri et al. [2003] which include terms for dynamical friction, hardening due to scattering of stars and gravitational wave emission.

$$\dot{a} = \min\left(-\frac{\rm G\rho_* a^2 H}{\sigma}, +\frac{2\dot{v}_{\rm DF} a}{v_c}\right) - \frac{256 G^3 M_{\bullet,1} M_{\bullet,2}(M_{\bullet,1}+M_{\bullet,2})}{5c^5 a^3} \tag{12.233}$$

where a is the black hole binary separation, H is a dimensionless hardening parameter $H \approx 15$ in the limit of a very hard, equal mass binary, $\rho_\star$ is the density of stars, $\dot{v}_{\rm DF}$ is the acceleration (negative) due to dynamical friction, $v_{\rm c}$ is the circular velocity, σ is the velocity dispersion of stars. Here the first factor represents hardening due to strong scattering of stars, the second results from dynamical friction with distant stars, gas and dark matter and the last results from the emission of gravitational waves Peters [1964].

The acceleration due to dynamical friction is computed using Chandrasekhar's formula:

$$\dot{v}_{\rm DF} = -\frac{2\pi {\rm G}^2 M_\bullet}{V_{\rm C}^2} \sum_i \rho_i \log(1+\Lambda_i^2) \left[{\rm erf}(X_i) - \left\{ \frac{2X_i}{\sqrt{\pi}} \exp\left(-X_i^2\right) \right\} \right], \tag{12.234}$$

where the sum is taken over the spheroid (gaseous plus stellar mass) and dark matter halo components[12]. Here,

$$\Lambda_i = \frac{a\sigma^2}{{\rm G}(M_{\bullet,1}+M_{\bullet,2})}, \tag{12.235}$$

is the Coulomb logarithm and

$$X_i = V_{\rm c}/\sqrt{2}\sigma. \tag{12.236}$$

In all of the above equations, the velocity dispersion σ_i is computed from the spherical Jeans equation assuming an isotropic velocity dispersion if `[blackHoleBinariesComputeVelocityDispersion]=true`. Otherwise, σ_i is set to the halo virial velocity for dark matter and to the spheroid characteristic velocity for the spheroid.

In calculating the rate of hardening due to scattering of stars, the stellar density is reduced by a factor [Volonteri et al., 2003]

$$f_\rho = \min\left\{ \left[\frac{4a\sigma^2_{\rm spheroid}}{3{\rm G}(M_{\bullet,1}+M_{\bullet,2})} \log\left(\frac{{\rm G}M_{\bullet,2}}{4\sigma^2_{\rm spheroid}a} \right) \right]^2, 1 \right\}, \tag{12.237}$$

if `[stellarDensityChangeBinaryMotion]=true` to account for the ejection of stars from the loss cone.

12.56. Supermassive Black Holes Binaries: Recoil Velocity

The method to be used for computing the recoil velocity due to gravitaional waves ejection during a binary merger specified by the `blackHoleBinaryRecoilVelocityMethod` parameter.

12.56.1. Null

Selected by default and with `blackHoleBinaryRecoilVelocityMethod=null`, this method assumes that there is no recoil velocity.

12.56.2. Campanelli et al. (2007)

Selected with `blackHoleBinaryRecoilVelocityMethod=Campanelli2007`, this method computes the recoil velocity during a black hole binary merger due to the emission of gravitational waves, following the formulae derived in Campanelli et al. [2007]

$$V_{\rm recoil} = V_{\rm m}\hat{\mathbf{e_1}} + V_\perp(\cos\xi\hat{\mathbf{e_1}} + \sin\xi\hat{\mathbf{e_2}}) + V_\parallel\hat{\mathbf{e_2}} \tag{12.238}$$

with:

$$V_{\rm m} = A\frac{q^2(1-q)}{(1+q)^5}(1+B\frac{q}{(1+q)^2}) \tag{12.239}$$

$$V_\perp = H\frac{q^2}{(1+q)^5}(\alpha_2^\parallel - q\alpha_1^\parallel) \tag{12.240}$$

[12]The disk is ignored as the black hole is assumed to be orbitting in a circular orbit in the disk.

$$V_{\parallel} = K\cos(\theta - \theta_0)\frac{q^2}{(1+q)^5}(\alpha_2^{\perp} - q\alpha_1^{\perp}) \tag{12.241}$$

where θ is defined as the angle between the inplane component of Δ and the infall direction at merger. q is the mass ratio of the black holes as $q = M_{\bullet,1}/M_{\bullet,2}$ and $\alpha_i = \mathbf{S_i}/M_{\bullet,i}$ depends of the spin and mass of the black hole ξ measures the angle between the unequal mass and the spin contribution to the recoil velocity in the orbital plane. $\hat{\mathbf{e_1}}, \hat{\mathbf{e_2}}$ are orthogonal unit vectors in the orbital plane. Our method assumes the spin of the second black hole is randomly generated, while that of the first is aligned with the angular momentum of the system. The constants used are retrieved from the articles by: Koppitz et al. [2007] for $H = (7.3 \pm 0.3)10^3$ km/s, González et al. [2007b] for $A = 1.2 \times 10^4$ km/s $B = -0.93$, González et al. [2007a] for $K\cos(\delta\theta) = (6, -5.3)10^4$ km/s and $K = (6.0 \pm 0.1)10^4$ km/s.

12.57. Supermassive Black Hole Binaries: Mergers

The method to be used for computing the effects of binary mergers of supermassive black holes is specified by the `blackHoleBinaryMergersMethod` parameter.

12.57.1. Rezzolla et al. (2008)

Selected with `blackHoleBinaryMergersMethod=Rezzolla2008`, this method uses the fitting function of Rezzolla et al. [2008] to compute the spin of the black hole resulting from a binary merger. The mass of the resulting black hole is assumed to equal the sum of the mass of the initial black holes (i.e. there is negligible energy loss through gravitational waves).

12.58. Tidal Induced Mass Loss Rates in Disks/Spheroids

The methods for computing tidal induced rates of mass loss in disks/spheroids are selected via the `tidalStrippingMassLossRate(Disks|Spheroids)Method` parameter.

12.58.1. Simple

Selected with `tidalStrippingMassLossRate(Disks|Spheroids)Method=simple` this method computes the mass loss rate to be:

$$\dot{M} = -\alpha M/\tau_{(\mathrm{disk|spheroid}}, \tag{12.242}$$

where

$$\alpha = \min(F_{\mathrm{tidal}}/F_{\mathrm{gravity}}, R_{\mathrm{maximum}}), \tag{12.243}$$

$F_{\mathrm{tidal}} = \mathcal{F}_{\mathrm{tidal}} r_{1/2}$, $\mathcal{F}_{\mathrm{tidal}}$ is the tidal field from the host halo (see §12.42),

$$F_{\mathrm{gravity}} = V_{1/2}^2(r_{1/2})/r_{1/2} \tag{12.244}$$

is the gravitational restoring force in the disk/spheroid at the half-mass radius, $r_{1/2}$, and $R_{\mathrm{maximum}} =$ `[tidalStrippingMassLos`

12.58.2. Null

Selected with `tidalStrippingMassLossRate(Disks|Spheroids)Method=null` this method assumes zero mass loss rate always.

12.59. Survey Geometry

The method to be used for computing the geometry of surveys for data analysis is specified by the `surveyGeometryMethod` parameter.

12.59.1. Li & White (2009)

Selected with `surveyGeometryMethod=liWhite2009SDSS`, this method describes the survey geometry of Li and White [2009].

For the angular mask, we make use of the catalog of random points within the survey footprint provided by the NYU-VAGC[13] (Blanton et al. 2005; see also Adelman-McCarthy et al. 2008, Padmanabhan et al. 2008). Li and White [2009] consider only the main, contiguous region and so we keep only those points which satisfy RA$>$ 100°, RA$<$ 300°, and RA$<$ 247° or $\delta < 51°$. When the survey window function is needed, these points are used to determine which elements of a 3D grid fall within the window function.

To estimate the depth of the Li and White [2009] sample as a function of galaxy stellar mass we make use of semi-analytic models in the Millennium Database. Specifically, we use the SAM of De Lucia and Blaizot (2007; specifically the `millimil..DeLucia2006a` and `millimil..DeLucia2006a_sdss2mass` tables in the Millennium Database). For each snapshot in the database, we extract the stellar masses and observed-frame SDSS r-band absolute magnitudes (including dust extinction), and determine the median absolute magnitude as a function of stellar mass. Using the limiting apparent magnitude of the Li and White [2009] sample, $r = 17.6$, we infer the corresponding absolute magnitude at each redshift and, using our derived absolute magnitude–stellar mass relation, infer the corresponding stellar mass.

The end result of this procedure is the limiting stellar mass as a function of redshift, accounting for k-corrections, evolution, and the effects of dust. Figure 12.1 shows the resulting relation between stellar mass and the maximum redshift at which such a galaxy would be included in the sample. Points indicate measurements from the SAM, while the line shows a polynomial fit:

$$\begin{aligned} z(M_\star) &= -5.950 + 2.638m - 0.4211m^2 \\ &\quad +2.852 \times 10^{-2}m^3 - 6.783 \times 10^{-4}m^4, \end{aligned} \tag{12.245}$$

where $m = \log_{10}(M_\star/M_\odot)$. We use this polynomial fit to determine the depth of the sample as a function of stellar mass. We adopt a solid angle of 2.1901993 sr [Percival et al., 2007] for the sample.

12.59.2. Bernardi et al. (2013)

Selected with `surveyGeometryMethod=bernardi2013SDSS`, this method describes the survey geometry of Bernardi et al. [2013].

For the angular mask, we make use of the MANGLE polygon file provided by the MANGLE project[14] The solid angle of this mask, computed using the MANGLE `harmonize` command is 2.232262776405 sr.

To determine the depth as a function of stellar mass, we make use of results provided by M. Bernardi (private communication), giving the mean maximum volume, $V_{\rm max}$, as a function of stellar mass for galaxies in this sample. These maximum volumes are converted to maximum distances using the solid angle quoted above. The results mass vs. distance relation is fit with a 5$^{\rm th}$-order polynomial. Figure 12.2 shows the resulting relation between stellar mass and the maximum distance at which such a galaxy would be included in the sample. Points indicate results from Bernardi, while the line shows a polynomial fit:

$$\log_{10}\left[\frac{D_{\rm max}(M_\star)}{\rm Mpc}\right] = 1282.11+m(-626.644+m(122.091+m(-11.8431+m(0.572399+m(-0.0110301))))) \tag{12.246}$$

[13]Specifically, http://sdss.physics.nyu.edu/lss/dr72/random/lss_random-0.dr72.dat.
[14]Specifically, http://space.mit.edu/ molly/mangle/download/data/sdss_dr72safe0_res6d.pol.gz.

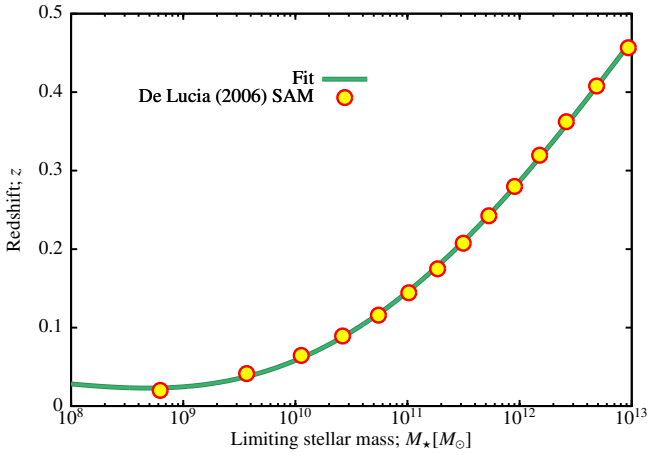

Figure 12.1.: The maximum redshift at which a galaxy of given stellar mass can be detected in the sample of Li and White [2009]. Points show the results obtained using the De Lucia and Blaizot [2007] model from the Millennium Database, while the lines shows a polynomial fit to these results (given in eqn. 12.245).

where $m = \log_{10}(M_\star/M_\odot)$. We use this polynomial fit to determine the depth of the sample as a function of stellar mass.

12.59.3. Davidzon et al. (2013)

Selected with `surveyGeometryMethod=davidzon2013VIPERS`, this method describes the survey geometry of Davidzon et al. [2013].

For the angular mask, we make use of `mangle` polygon files provided by I. Davidzon (private communication) corresponding to the VIPERS fields. The solid angle of each mask is computed using the `mangle harmonize` command.

To determine the depth as a function of stellar mass, we make use of the tabulated mass function, ϕ, and number of galaxies per bin, N, supplied by I. Davidzon (private communication). The effective volume of each bin is found as $V_i = N_i/f_{\rm complete}\phi_i\Delta\log_{10} M_\star$, where $\Delta\log_{10} M_\star$ is the width of the bin, and $f_{\rm complete}$ is the completeness of the survey, estimated to be approximately 40% [Guzzo et al., 2013]. These volumes are converted to maximum distances in each field using the survey solid angle. The resulting mass vs. distance relation in each field is fit with a 1$^{\rm st}$-order polynomial in log-log space over the range where the maximum volume is limited by the survey depth and not by the imposed upper limit to redshift. Figure 12.3 shows the resulting relation between stellar mass and the maximum distance at which such a galaxy would be included in the sample. Points indicate results from VIPERS, while the lines show polynomial fits:

$$\log_{10}\left[\frac{D_{\rm max}(M_\star)}{\rm Mpc}\right] = \begin{cases} 3.207 + 0.0124m & 0.5 < z < 0.6 \\ 3.148 + 0.0268m & 0.6 < z < 0.8 \\ 3.207 + 0.0273m & 0.8 < z < 1.0 \end{cases} \tag{12.247}$$

where $m = \log_{10}(M_\star/M_\odot)$. We use this polynomial fit to determine the depth of the sample as a function of stellar mass.

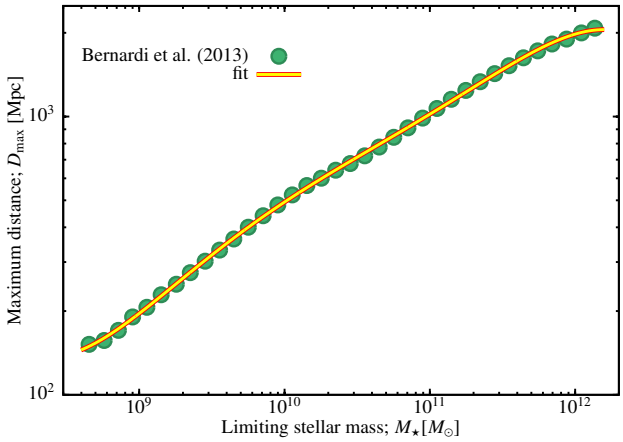

Figure 12.2.: The maximum distance at which a galaxy of given stellar mass can be detected in the sample of Bernardi et al. [2013]. Points show the results obtained from data provided by Bernardi, while the lines shows a polynomial fit to these results (given in eqn. 12.246).

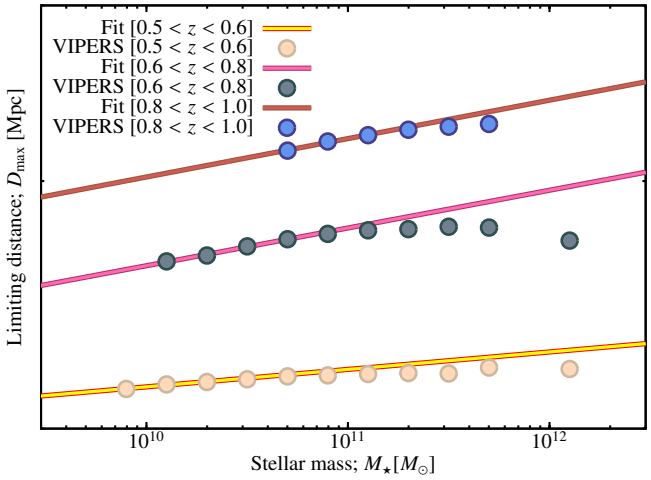

Figure 12.3.: The maximum distance at which a galaxy of given stellar mass can be detected in the sample of Davidzon et al. [2013]. Points show the results obtained from data provided by Davidzon, while the lines shows a polynomial fit to these results (given in eqn. 12.247). Note that at high masses the distance is limited by the imposed upper limit—the polynomial fit does not consider these points.

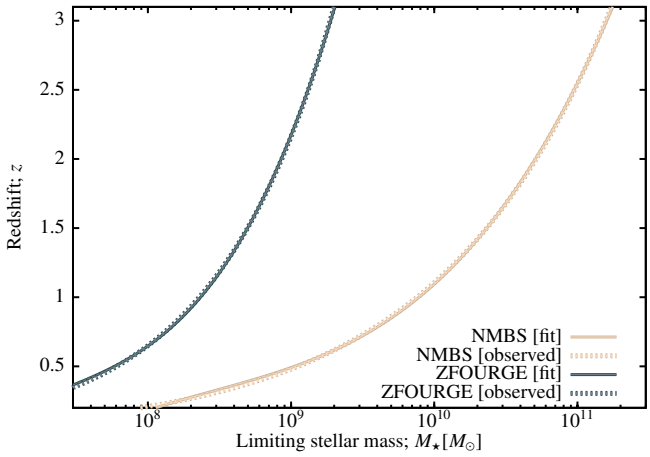

Figure 12.4.: The maximum redshift at which a galaxy of given stellar mass can be detected in the sample of Tomczak et al. [2014]. Points show the results obtained from data provided by Davidzon, while the lines shows a polynomial fit to these results (given in eqn. 12.248).

12.59.4. Tomczak et al. (2014)

Selected with `surveyGeometryMethod=tomczak2014ZFOURGE`, this method describes the survey geometry of Tomczak et al. [2014].

For the angular mask, we make use of `mangle` polygon files constructed by hand using vertices matched approximately to the distribution of galaxies in the survey (positions of which were provided by R. Quadri; private communication). The solid angle of each mask is computed using the `mangle harmonize` command.

To determine the depth as a function of stellar mass, we make use of the tabulated mass completeness limits as a function of redshift for ZFOURGE and NMBS fields provided by R. Quadri (private communication). These are fit with fourth-order polynomials. Figure 12.4 shows the resulting relation between stellar mass and the maximum redshift at which such a galaxy would be included in the sample. Dotted lines indicate the tabulated result from ZFOURGE, while the lines show polynomial fits:

$$z_{\max}(M_\star) = \begin{cases} -114.66 + m*(45.901 + m*(-6.1617 + m*(0.27822))) & \text{ZFOURGE fields} \\ -58.483 + m*(20.250 + m*(-2.3563 + m*(0.092705))) & \text{NMBS fields} \end{cases} \tag{12.248}$$

where $m = \log_{10}(M_\star/M_\odot)$. We use this polynomial fit to determine the depth of the sample as a function of stellar mass.

12.59.5. Baldry et al. (2012)

Selected with `surveyGeometryMethod=baldry2012GAMA`, this method describes the survey geometry of Baldry et al. [2012].

For the angular mask we use the specifications of the G09, G12, and G15 fields given by Driver et al. [2011] to construct MANGLE polygon files.

To determine the depth as a function of stellar mass, we make use of the publicly available tabulated mass function, ϕ, and number of galaxies per bin, N. The effective volume of each bin is found as $V_i = N_i/\phi_i \Delta \log_{10} M_\star$, where $\Delta \log_{10} M_\star$ is the width of the bin. The GAMA survey consists of three fields, each of the same solid angle, but with differing depths. We assume that the relative depths in

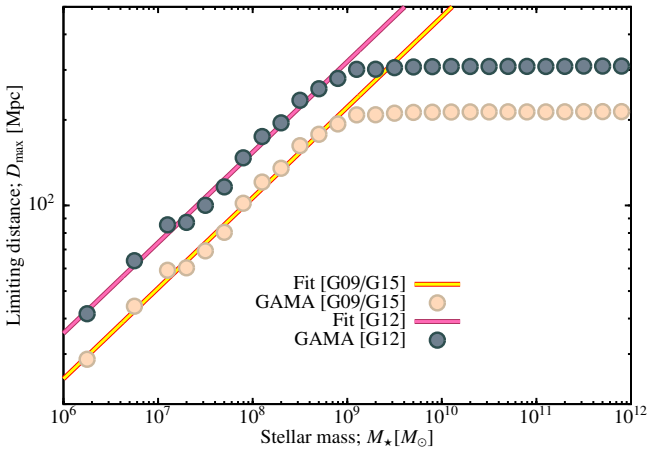

Figure 12.5.: The maximum distance at which a galaxy of given stellar mass can be detected in the sample of Baldry et al. [2012]. Points show the results obtained from data provided by Baldry, while the lines shows a polynomial fit to these results (given in eqn. 12.249). Note that above $10^9 M_\odot$ the distance is limited by the imposed upper limit of $z = 0.06$ in the GAMA sample—the polynomial fit does not consider these points.

terms of stellar mass scale with the depth in terms of flux. Given this assumption, these volumes are converted to maximum distances in each field using the solid angle quoted above. The resulting mass vs. distance relation in each field is fit with a 1$^{\rm st}$-order polynomial in log-log space over the range where the maximum volume is limited by the survey depth and not by the imposed $z = 0.06$ upper limit to redshift. Figure 12.5 shows the resulting relation between stellar mass and the maximum distance at which such a galaxy would be included in the sample. Points indicate results from GAMA, while the line shows a polynomial fit:

$$\log_{10}\left[\frac{D_{\rm max}(M_\star)}{\rm Mpc}\right] = \begin{cases} -0.521 + 0.319m & \text{fields G09/G15} \\ -0.361 + 0.319m & \text{field G12} \end{cases} \tag{12.249}$$

where $m = \log_{10}(M_\star/M_\odot)$. We use this polynomial fit to determine the depth of the sample as a function of stellar mass.

12.59.6. Moustakas et al. (2013)

Selected with `surveyGeometryMethod=moustakas2013PRIMUS`, this method describes the survey geometry of Moustakas et al. [2013].

For the angular mask, we make use of MANGLE polygon files provided by J. Moustakas (private communication) corresponding to the give PRIMUS fields. The solid angle of each mask is computed using the MANGLE `harmonize` command.

To determine the depth as a function of stellar mass, we make use of completeness limits for "All" galaxies given in Table 2 of Moustakas et al. [2013]. These are fit, for each field with a second order polynomial to give the limiting redshift as a function of stellar mass. Figure 12.6 shows the resulting relation between stellar mass and the maximum redshift at which such a galaxy would be included in the

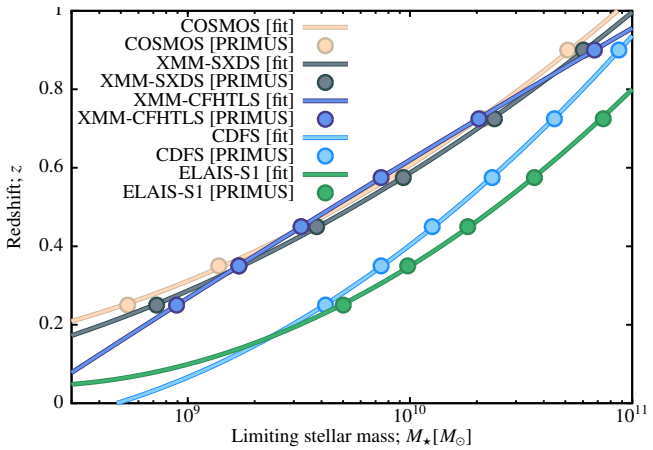

Figure 12.6.: The maximum distance at which a galaxy of given stellar mass can be detected in the sample of Moustakas et al. [2013]. Points show the results obtained from completeness limit data taken from Table 2 of Moustakas et al. [2013], while the lines shows a polynomial fit to these results (given in eqn. 12.254).

sample. Points indicate results from Moustakas et al. [2013], while the line shows a polynomial fits:

$$z_{\max}(M_\star) = +3.51 + m(-0.941 + m(+0.0651)) \quad \text{COSMOS} \tag{12.250}$$
$$z_{\max}(M_\star) = +2.46 + m(-0.730 + m(+0.0542)) \quad \text{XMM-SXDS} \tag{12.251}$$
$$z_{\max}(M_\star) = -3.60 + m(+0.500 + m(-0.0078)) \quad \text{XMM-CFHTLS} \tag{12.252}$$
$$z_{\max}(M_\star) = +5.87 + m(-1.528 + m(+0.0982)) \quad \text{CDFS} \tag{12.253}$$
$$z_{\max}(M_\star) = +6.87 + m(-1.656 + m(+0.1003)) \quad \text{ELAIS-S1} \tag{12.254}$$

where $m = \log_{10}(M_\star/M_\odot)$. We use this polynomial fit to determine the depth of the sample as a function of stellar mass.

12.59.7. Muzzin et al. (2013)

Selected with `surveyGeometryMethod=muzzin2013ULTRAVISTA`, this method describes the survey geometry of Muzzin et al. [2013].

For the angular mask, we generate a MANGLE polygon file, by first defining a rectangle encompassing the bounds of the ULTAVISTA field ($149.373° < \alpha < 150.779°$ and $1.604° < \delta < 2.81°$). From this rectangle, we then remove circles of radii $75''$ around bright stars (i.e. those bright than 10^{th} and 8^{th} magnitudes in the USNO and 2MASS star lists respectively) and radii $30''$ around medium stars (i.e. those bright than 13^{th} and 10.5^{th} magnitudes in the USNO and 2MASS star lists respectively). Finally, we mask regions of one detector for which 75% of pixels are dead by clipping pixels with weights below 0.02 in the K_s-band weight map. These choices match those made in the ULTRAVISTA survey (A. Muzzin, private communication). The solid angle of each mask is computed using the MANGLE `harmonize` command.

To determine the depth as a function of stellar mass, we simply fit the tabulated relations provided by the ULTRAVISTA survey:

$$z_{\max}(M_\star) = \frac{-8364.45 + m(4331.82 + m(-896.596 + m(92.6999 + m(-4.78750 + m(0.0988215)))))}{1 - \exp[(m - 11.24)/0.02]} \tag{12.255}$$

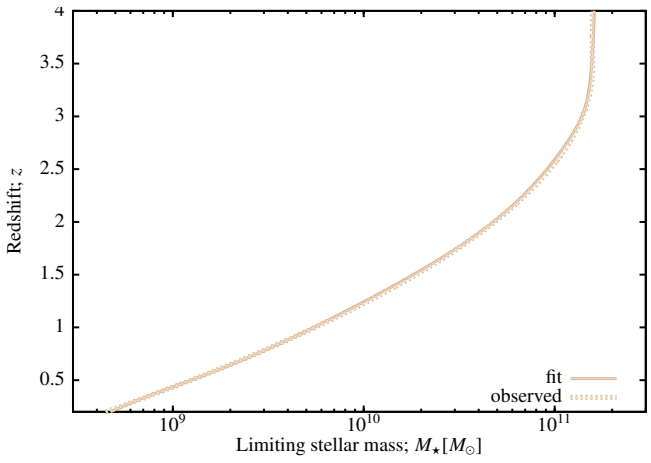

Figure 12.7.: The maximum distance at which a galaxy of given stellar mass can be detected in the sample of Muzzin et al. [2013]. The dotted line shows the results obtained from the ULTRAVISTA survey [Muzzin et al., 2013], while the solid line shows the polynomial fit to these results (given in eqn. 12.255).

where $m = \log_{10}(M_\star/M_\odot)$.

12.59.8. Martin et al. (2010)

Selected with `surveyGeometryMethod=martin2010ALFALFA`, this method describes the survey geometry of Martin et al. [2010].

For the angular mask we use the three disjoint regions defined by $07^{\rm h}30^{\rm m} <$ R.A. $< 16^{\rm h}30^{\rm m}$, $+04° <$ decl. $< +16°$, and $+24° <$ decl. $< +28°$ and $22^{\rm h} <$ R.A. $< 03^{\rm h}$, $+14° <$ decl. $< +16°$, and $+24° <$ decl. $< +32°$ corresponding to the sample of Martin et al. [2010]. When the survey window function is needed we generate randomly distributed points within this angular mask and out to the survey depth. These points are used to determine which elements of a 3D grid fall within the window function.

To estimate the depth of the Martin et al. [2010] sample as a function of galaxy HI mass we first infer the median line width corresponding to that mass. To do so, we have fit the median line width-mass relation from the $\alpha.40$ sample with power-law function as shown in Fig. 12.8. We find that the median line width can be approximated by

$$\log_{10}(W_{50}/\text{km s}^{-1}) = c_0 + c_1 \log_1 0(M_{\rm HI}/M_\odot), \tag{12.256}$$

with $c_0 = -0.770$ and $c_1 = 0.315$. Given the line width, the corresponding integrated flux limit, $S_{\rm int}$, for a signal-to-noise of 6.5 is inferred using equation (A1) of Haynes et al. [2011]. Finally, this integrated flux limit is converted to maximum distance at which the source could be detected using the expression given in the text of section 2.2 of Martin et al. [2010]:

$$M_{\rm HI} = 2.356 \times 10^5 \left(\frac{D}{\text{Mpc}}\right)^2 \left(\frac{S_{\rm int}}{\text{Jy km s}^{-1}}\right). \tag{12.257}$$

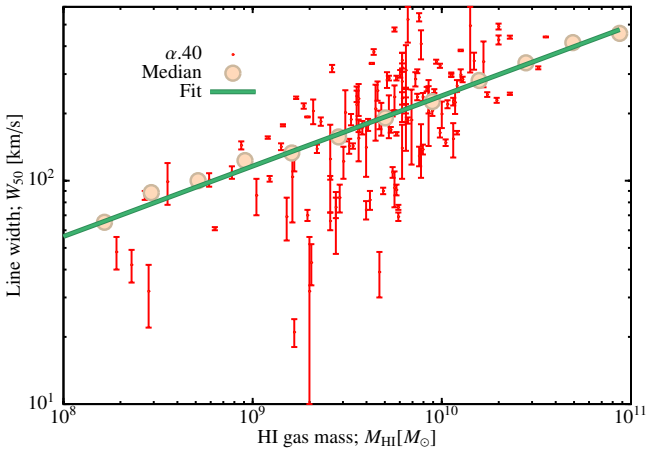

Figure 12.8.: HI line width vs. HI mass as measured from the $\alpha.40$ survey of Martin et al. [2010]. Red points with error bars show individual measurements, while the larger circles indicate the running median of these data. The green line is a power-law fit to the running median as described in eqn. (12.256).

12.59.9. Caputi et al. (2011)

Selected with `surveyGeometryMethod=caputi2011UKIDSSUDS`, this method describes the survey geometry of Caputi et al. [2011].

The angular mask is defined by the boundaries given in Table 12.4 together with a set of rectangular holes fitted to gaps in the galaxy distribution provided by Karina Caputi (private communication). In particular, a set of points randomly distributed in the survey mask is constructed by the script `constraints/dataAnalysis/stellarMassFunctions_UKIDSS_UDS_z3_5/surveyGeometryRandoms.pl`. These points are used to determine which elements of a 3D grid fall within the window function. We compute a solid angle of 1.592×10^{-4} sr for the sample from this window function.

To estimate the depth of the Caputi et al. [2011] sample as a function of galaxy stellar mass we make use of semi-analytic models in the Millennium Database. Specifically, we use the SAMs of Henriques et al. [2012] and Guo et al. (2011; specifically the `Henriques2012a.wmap1.BC03_001` and `Guo2010a..MR` tables in the Millennium Database). For each snapshot in the database, we extract the stellar masses and observed-frame IRAC 4.5μm apparent magnitudes (including dust extinction), and determine the median apparent magnitude as a function of stellar mass. Using the limiting apparent magnitude of the Caputi et al. [2011] sample, 24.0, we infer the corresponding stellar mass.

The end result of this procedure is the limiting stellar mass as a function of redshift, accounting for k-corrections, evolution, and the effects of dust. Figure 12.9 shows the resulting relation between stellar mass and the maximum redshift at which such a galaxy would be included in the sample. Points indicate measurements from the SAM, while the line shows a polynomial fit:

$$z(M_\star) \quad = \quad -56.247 + 5.881 \log_{10}(M_\star/M_\odot). \tag{12.258}$$

We use this polynomial fit to determine the depth of the sample as a function of stellar mass.

Table 12.4.: Boundaries of the angular mask for the survey of Caputi et al. [2011]. Each boundary consists of a great circle segment defined by start and end coordinates (given in tuples of right acesension and declination in units of degrees), together with a "type" which specifies whether points above, below, to the right or left of the boundary are part of the survey (where "above" and "below" refer to declination, and "left" and "right" refer to right acsension).

Start	End	Type
(34.460,-4.648)	(34.750,-4.648)	above
(34.000,-4.908)	(34.460,-4.648)	above
(34.900,-4.920)	(34.750,-4.648)	above
(34.440,-5.518)	(34.905,-5.255)	below
(34.210,-5.518)	(34.440,-5.518)	below
(34.210,-5.518)	(34.000,-5.150)	below
(34.000,-4.908)	(34.000,-5.150)	left
(34.900,-4.920)	(34.905,-5.255)	right

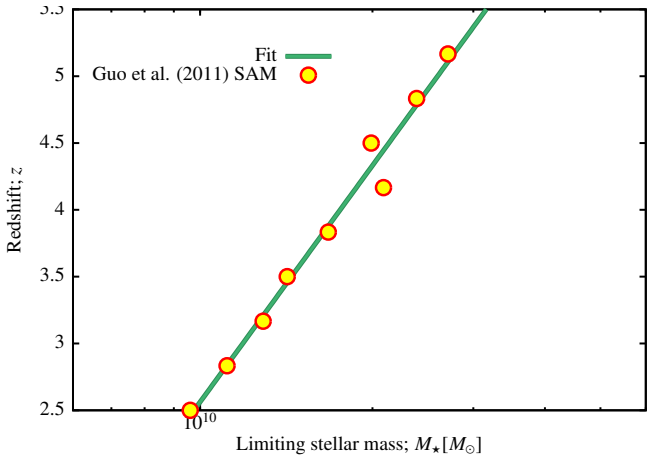

Figure 12.9.: The maximum redshift at which a galaxy of given stellar mass can be detected in the sample of Caputi et al. [2011]. Points show the results obtained using the Henriques et al. [2012] and Guo et al. [2011] models from the Millennium Database, while the lines shows a polynomial fit to these results (given in eqn. 12.258).

13. Additional Output Quantities

13.1. Black Hole Accretion

Properties associated with accretion onto supermassive black holes can be output by setting `blackHoleOutputAccretion=tru`
Currently, two additional properties are output for each node when this option is selected:

`blackHoleAccretionRate` The rate at which the supermassive black hole is accreting mass in $M_\odot$ Gyr^{-1};

`blackHoleJetPower` The power being emitted into jets by the black hole/accretion disk system in $M_\odot$ km^2 s^{-2} Gyr^{-1}.

13.2. Cooling Data

Properties associated with cooling in hot halos can be output by setting `hotHaloOutputCooling=true`.
Currently, two additional properties are output for each node when this option is selected:

`hotHaloCoolingRate` The rate at which gas is cooling from the halo (assuming no sources of heating) in $M_\odot$ Gyr^{-1};

`hotHaloCoolingRadius` The characteristic cooling radius in the halo in Mpc.

13.3. Density Contrast Data

Properties of nodes at density contrasts other than the virial density can be output by setting `[outputDensityContrastData]=`
When selected, this output option requires that a list of density contrasts, Δ (defined in units of the mean density of the Universe), be given in the `[outputDensityContrastValues]` input parameter. For each specified density contrast, two properties are output for each node: `nodeRadius`Δ and `nodeMass`Δ which give the radius enclosing a mean density contrast of Δ and the mass enclosed within that radius. The parameter `[outputDensityContrastDataDarkOnly]` controls whether density contrasts are measured for total mass (`false`) or dark matter mass only (`true`). In the latter case, density contrasts are defined relative to the mean dark matter density of the Universe.

13.4. Descendent Node Index

By setting `[outputDescendentIndices]=true` the index of the node containing the galaxy to which each current galaxy will belong at the next output time (i.e. the forward descendent) will be written to the output file. To clarify, this will be the index of the node into which the galaxy descends, or the index of a node with which it merges prior to the next output time (and if that node merges with another, the index will be of that node and so on).

Note that, to operate correctly, information about which node a given node may merge with (and when this merger will happen) must be available. This is typically available in merger trees read from file providing `[treeNodeMethodSatelliteOrbit]` and `[mergerTreeReadPresetMergerTimes]` are both set to `true`. When using randomly assigned satellite orbits and merger times, information on when merging

occurs does not exist until a node becomes a satellite. Thus, if the node becomes a satellite after the current output, but before the next output, there is no way to know which node it will belong to at the next output (in such cases, the fallback assumption is no merging).

13.5. Host Node Index

By setting `[outputHostIndices]=true` the index of the node which hosts each node will be output to the `hostIndex` dataset. For unhosted nodes (i.e. nodes which are not subhalos), a value of -1 is output instead.

13.6. Half-Light Radii Data

Half-light radii and masses enclosed within them can be output by setting `[outputHalfLightData]=true`. When selected the half-light radius in each specified luminosity band is output as `[halfLightRadius{luminosityID}]` (in Mpc), where `{luminosityID}` is the usual luminosity identifier suffix, and the total (dark + baryonic) mass within that radius is output as `[halfLightMass{luminosityID}]` (in $M_\odot$).

13.7. Half-Mass Radii

Half-mass radii can be output by setting `[outputHalfMassData]=true`. When selected the half-mass radius is output as `[halfMassRadius]` (in Mpc).

13.8. Halo Model Quantities

The following quantities related to galaxy clustering are output if `[outputHaloModelData]` is set to true:

`nodeBias` The large scale, lineary theory bias for each node. For satellite nodes, this corresponds to the bias of their host halo;

`isolatedHostIndex` The index of the isolated node in which this node lives. This is identical to `nodeIndex` for non-satellite nodes.

In addition to these quantities output for each node, setting `[outputHaloModelData]=true` causes the creation of a `haloModel` group in the Galacticus output file. This group contains the following:

`wavenumber` A dataset giving the wavenumbers (in units of Mpc^{-1}) at which all output power spectra are tabulated. The minimum and maximum wavenumbers to tabulate are determined by the `[haloModelWavenumberMinimum]` and `[haloModelWavenumberMaximum]` parameters respectively, while the number of points to tabulate in each decade of wavenumber is determined by the `[haloModelWavenumberPointsPerDecade]` parameter.

`powerSpectrum` A dataset giving the linear theory power spectrum (in units of Mpc^3 normalized to $z = 0$ at each wavenumber specified in the `wavenumber` dataset.

`Output{i}/mergerTree{j}/fourierProfile{k}` A dataset giving the Fourier transform of the dark matter halo density profile (dimensionless and normalized to unity at small wavenumber) for the node with index `k` in merger tree with index `j` at output number `i`. Profiles are written only for nodes which are isolated, and are tabulated at the wavenumbers given in the `wavenumber` group. Note that wavenumbers are assumed to be comoving.

Finally, each numbered output group is given two additional attributes, `linearGrowthFactor` and `linearGrowthFactorLogDerivative` which give the growth factor, D, and its logarithmic derivative, $\mathrm{d}\ln D/\mathrm{d}\ln a$ at the output time.

The information output can be used to construct galaxy power spectra and correlation functions.

13.9. Lightcone Coordinates

The position (and velocity and redshift) of a galaxy within a lightcone will be output if the parameter `[outputLightconeData]=true`. In such cases, the following properties will be output for all galaxies:

`lightconePositionX` Position of the galaxy (in comoving Mpc) along the radial direction of the lightcone;

`lightconePositionY` Position of the galaxy (in comoving Mpc) along the 1$^{\mathrm{st}}$ angular direction of the lightcone;

`lightconePositionZ` Position of the galaxy (in comoving Mpc) along the 2$^{\mathrm{nd}}$ angular direction of the lightcone;

`lightconeVelocityX` Velocity of the galaxy (in km/s) along the radial direction of the lightcone;

`lightconeVelocityY` Velocity of the galaxy (in km/s) along the 1$^{\mathrm{st}}$ angular direction of the lightcone;

`lightconeVelocityZ` Velocity of the galaxy (in km/s) along the 2$^{\mathrm{nd}}$ angular direction of the lightcone;

`lightconeRedshift` Redshift of the galaxy in the lightcone[1];

`angularWeight` The mean number density of this galaxy per unit area on the sky (in degrees^{-2}).

In order to allow this output a lightcone geometry must be specified, see §16.4.1.

13.10. Main Branch Evolution

The evolution of main branch galaxies can be recorded by setting `[timestepRecordEvolution]=true`. When set, the evolution of each main branch galaxy will be recorded at a set of `[timestepRecordEvolutionSteps]` timesteps spaced logarithmically in cosmic time between `[timestepRecordEvolutionBegin]` and `[timestepRecordEvolutionEnd]`.

This recorded evolution will be written to the group `mainProgenitorEvolution` in the Galacticus output file. Within that group two datasets, `time` and `expansionFactor`, give the times and expansion factors at which evolution was recorded. Then for each merger tree two datasets, `stellarMass<N>` and `totalMass<N>` (where `<N>` is the merger tree index), give the stellar and total baryonic mass of the main branch progenitor at each timestep.

13.11. Main Branch Status

The status of each node with respect to the main branch of its merger tree can be output by setting `[outputMainBranchStatus]=true`. When set, the status will be output as `nodeIsOnMainBranch`, with a value of 1 indicating that the node is a primary progenitor of the final halo (i.e. is on the main branch of the tree) and a value of 0 indicating that it is not.

[1] Note that this will not, in general, be precisely the same as the redshift corresponding to the output time.

13.12. Mass Profile Data

Masses enclosed within specific radii can be output by setting `[outputMassProfileData]=true`. When selected, this output option requires that a list of radii, r (in Mpc), be given in the `[outputMassProfileRadii]` input parameter. For each specified radius, the total (dark + baryonic) mass will be output as `massProfile`r.

13.13. Merger Tree Links and Node Isolation

The following properties are output to permit the merger tree structure to be recovered:

`nodeIndex` A unique[2] (within a tree) integer index identifying the node;

`parentIndex` The index of this node's parent node (or -1 if it has no parent);

`siblingIndex` The index of this node's sibling node (or -1 if it has no sibling);

`satelliteIndex` The index of this node's first satellite node (or -1 if it has no satellites);

`nodeIsIsolated` Will be 0 for a node which is a subhalo inside some other node (i.e. a satellite galaxy) or 1 for a node that is an isolated halo (i.e. a central galaxy).

The `nodeIndex` property corresponds by default to the index of the node in the original merger tree. This means that as a galaxy evovles through the tree and, in particular, gets promoted into a new halo the index associated with a galaxy will change. This is useful to identify where the galaxy resides in the original (unevolved) tree structure, but does not allow galaxies to be traced from one output to the next using their `nodeIndex` value. By setting `[nodePromotionIndexShift]=true` this behavior can be changed such that the value of `nodeIndex` will reflect the index of the earliest progenitor node along the main branch of the current node. As such, this index will remain the same for a given galaxy during its evolution. These two alternative algorithms for propagating node indices are illustrated in Figure 13.1.

13.14. Merger Tree Data for Rendering

Data on the structure of a merger tree and its halos useful for rendering the tree as a 3-D structure can be output using the `Merger_Trees_Render` module. Calling `Merger_Trees_Render_Dump` with a tree as the only argument will cause the tree structure will be dumped to a file named `render_`⟨`treeIndex`⟩`_`⟨`outputIndex`⟩`.hdf5` where ⟨`treeIndex`⟩ is the index of the tree and ⟨`outputIndex`⟩ is an incremental counter that tracks the number of outputs for this tree. The output is a simple HDF5 file containing the following datasets:

`nodeIndex` Index of the node;

`parentIndex` Index of the parent node;

`childIndex` Index of the child node;

`time` Time of the node;

`expansionFactor` Corresponding expansion factor;

`radiusVirial` Virial radius of the node;

`position` (x, y, z) position of the node.

[2] Node indices are typically unique, but there is no actual requirement within Galacticus that this must be the case. A merger tree construction method could create nodes with non-unique indices.

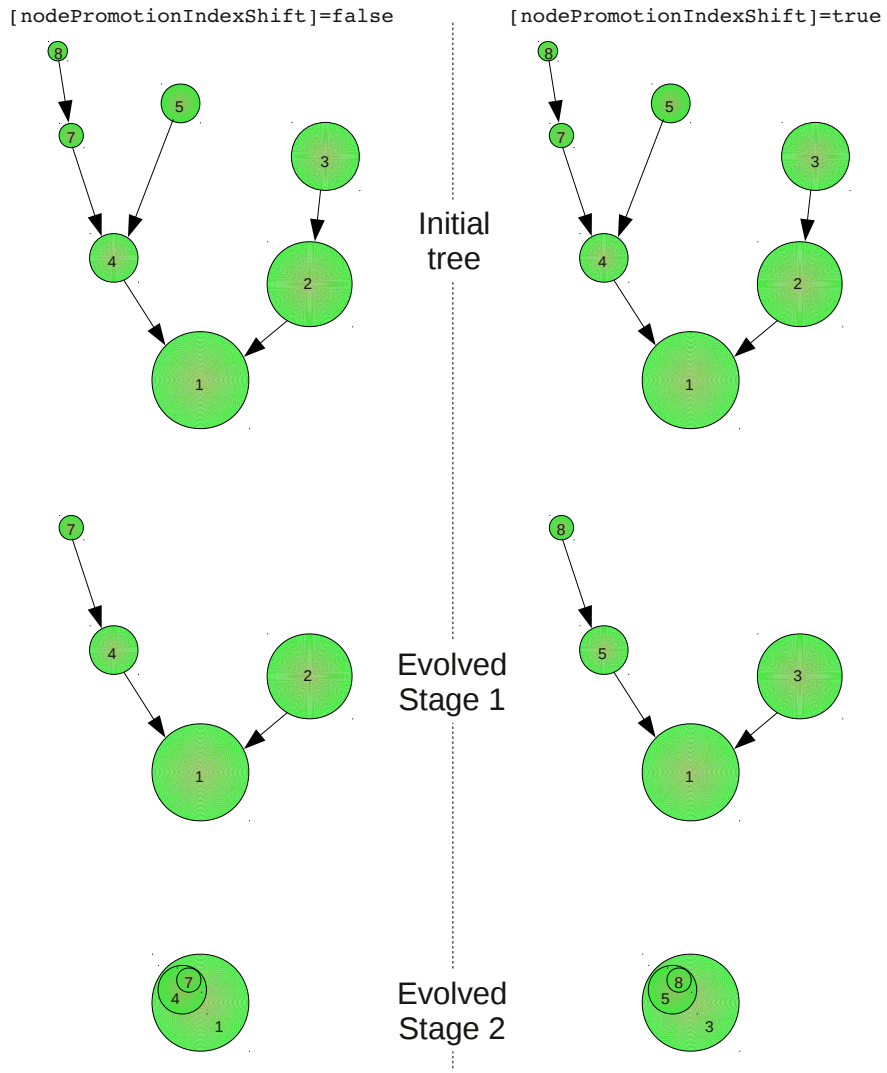

Figure 13.1.: Illustration of options for the propagation of node indices during node promotion events. Two identical trees (top row) are evolved with `[nodePromotionIndexShift]=false` (left column) and `[nodePromotionIndexShift]=true` (right column). The middle and lower rows indicate the resulting node indices after two stages of tree evolution.

13.15. Merger Tree Structure

The structure of each merger tree can optionally be dumped to a file suitable for post-processing with DOT after every step of evolution. To request this output set `[mergerTreesDumpStructure]=true`. After each evolution step the tree structure will be dumped to a file named `mergerTreeDump:⟨treeIndex⟩:⟨outputIndex⟩.gv` where ⟨`treeIndex`⟩ is the index of the tree and ⟨`outputIndex`⟩ is an incremental counter that tracks the number of outputs for this tree. These files can be processed with DOT to produce a diagram of the tree structure. The node currently being evolved will be highlighted in green. This output option makes use of the `Merger_Trees_Dump` module to create the outputs.

13.16. Most Massive Progenitor

Setting `[outputMostMassiveProgenitor]=true` causes the property `isMostMassiveProgenitor` to be output. This property will be 1 for the most massive progenitor node in a tree at each output time and 0 for all other nodes.

13.17. Projected Density

Setting `[outputProjectedDensityData]=true` causes the projected density of the node to be output at specified radii. The radii and types of projected density to output is specified by the `outputProjectedDensityRadii` parameter. This parameter's value can contain multiple entries. Each entry is of the form:

```
radiusType:componentType:massType:loading?:radius
```

The elements of this colon-separated specifier determine the radius at which the projected density is computed, which components/mass types should be counted, and whether baryonic loading of the halo should be accounted for. The elements have the following meaning:

`radius` the numerical value of the radius at which to compute the projected density (with units specified by the `radiusType` element);

`radiusType` specifies the units of the `radius` element—valid options are `diskRadius`, `diskHalfMassRadius`, `spheroidRadius`, `spheroidHalfMassRadius`, `darkMatterScaleRadius`, `virialRadius`, just `radius` (which implies radii are given in units of Mpc), `galacticMassFraction{<fraction>}`, or `galacticLightFraction{<fraction>}{<luminosity>}`, where the final two form specify a radius containing a fixed `<fraction>` of the galactic mass or light respectively (for the case of galactic light, `<luminosity>` specifies the band, e.g. `SDSS_r:rest:z0.0000`);

`componentType` specifies which components of the node should be counted—allowed values are `all`, `disk`, `spheroid`, `hotHalo`, `darkHalo`, and `blackHole`;

`massType` specifies which types of mass should be counted—allowed values are `all`, `dark`, `baryonic`, `galactic`, `gaseous`, `stellar`, and `blackHole`;

`loading?` option should be either `loaded` or `unloaded`, and specifies whether the effect of baryonic loading (i.e. adiabatic contraction) should be included in the calculation of the projected density.

13.18. Rotation Curve

Setting `[outputRotationCurveData]=true` causes the rotation curve of the node to be output at specified radii. The radii and types of rotation curve to output is specified by the `outputRotationCurveRadii` parameter. This parameter's value can contain multiple entries. Each entry is of the form:

```
radiusType:componentType:massType:loading?:radius
```

The elements of this colon-separated specifier determine the radius at which the rotation curve is computed, which components/mass types should be counted, and whether baryonic loading of the halo should be accounted for. The elements have the following meaning:

`radius` the numerical value of the radius at which to compute the rotation curve (with units specified by the `radiusType` element);

`radiusType` specifies the units of the `radius` element—valid options are `diskRadius`, `diskHalfMassRadius`, `spheroidRadius`, `spheroidHalfMassRadius`, `darkMatterScaleRadius`, `virialRadius`, just `radius` (which implies radii are given in units of Mpc), `galacticMassFraction{<fraction>}`, or `galacticLightFraction{<fraction>}{<luminosity>}`, where the final two form specify a radius containing a fixed `<fraction>` of the galactic mass or light respectively (for the case of galactic light, `<luminosity>` specifies the band, e.g. `SDSS_r:rest:z0.0000`);

`componentType` specifies which components of the node should be counted—allowed values are `all`, `disk`, `spheroid`, `hotHalo`, `darkHalo`, and `blackHole`;

`massType` specifies which types of mass should be counted—allowed values are `all`, `dark`, `baryonic`, `galactic`, `gaseous`, `stellar`, and `blackHole`;

`loading?` option should be either `loaded` or `unloaded`, and specifies whether the effect of baryonic loading (i.e. adiabatic contraction) should be included in the calculation of the rotation curve.

13.19. Satellite Mergers

Setting `[outputSatelliteMergers]` to `true` will cause data to be output for each satellite merger event. Data is stored to datasets in the `satelliteMergers` group. Datasets output are:

`[indexHost]` the index of the host halo in the merger;

`[indexSatellite]` the index of the satellite halo in the merger;

`[indexTree]` the index of the merger tree in which the merger occurred;

`[massHost]` the mass of the host halo;

`[massSatellite]` the mass of the satellite halo;

`[time]` the time at which the merger occurred.

13.20. Satellite Orbital Pericenter

Setting `[outputSatellitePericenterData]` to `true` will cause the pericentric values of the radius and velocity of each satellite node's orbit to be output as `satellitePericenterRadius` and `satellitePericenterVelocity` respectively.

13.21. Star Formation Rates

By default the star formation rate in each galaxy is *not* output. However, setting `[diskOutputStarFormationRate]` to true will cause the current star formation rate in the disk of each galaxy to be output as `diskStarFormationRate` (in units of $M_\odot$/Gyr). The `[spheroidOutputStarFormationRate]` has the same effect for the spheroid component.

13.22. Velocity Dispersion

Setting `[outputVelocityDispersionData]=true` causes the velocity dispersion of the node to be output at specified radii, for specified components. The radii, components and type of velocity dispersion to output are specified by the `outputVelocityDispersionRadii` parameter. This parameter's value can contain multiple entries. Each entry is of the form:

```
radiusType:componentType:massType:loading?:direction:radius
```

The elements of this colon-separated specifier determine the radius at which the velocity dispersion is computed, which component/mass type the velocity dispersion should be computed for, whether baryonic loading of the halo should be accounted for, and for which direction the velocity dispersion should be computed. The elements have the following meaning:

`radius` the numerical value of the radius at which to compute the velocity dispersion (with units specified by the `radiusType` element);

`radiusType` specifies the units of the `radius` element—valid options are `diskRadius`, `diskHalfMassRadius`, `spheroidRadius`, `spheroidHalfMassRadius`, `darkMatterScaleRadius`, `virialRadius`, just `radius` (which implies radii are given in units of Mpc), `galacticMassFraction{<fraction>}`, or `galacticLightFraction{<fraction>}{<luminosity>}`, where the final two form specify a radius containing a fixed `<fraction>` of the galactic mass or light respectively (for the case of galactic light, `<luminosity>` specifies the band, e.g. `SDSS_r:rest:z0.0000`);

`componentType` specifies which component of the node the velocity dispersion should be computed for—allowed values are `all`, `disk`, `spheroid`, `hotHalo`, `darkHalo`, and `blackHole`[3];

`massType` specifies which type of mass the velocity dispersion should be computed for—allowed values are `all`, `dark`, `baryonic`, `galactic`, `gaseous`, `stellar`, and `blackHole`;

`loading?` option should be either `loaded` or `unloaded`, and specifies whether the effect of baryonic loading (i.e. adiabatic contraction) should be included in the calculation of the velocity dispersion;

`direction` should be one of `radial` (computes the radial component of velocity dispersion), `lineOfSight{<luminosity>}` (computes the line-of-sight velocity dispersion), `lineOfSightInteriorAverage{<luminosity>}` (computes the line-of-sight velocity dispersion averaged interior to the given radius), or `lambdaR{<luminosity>}` (computes the λ_R statistic of Cappellari et al. 2007)—in the latter three cases `{<luminosity>}` specifies which band should be used to weight the velocity dispersion, alternatively setting `{<luminosity>}=mass` (or just leaving off this specifier entirely) will use mass weighting instead.

13.23. Virial Quantities

The following quantities related to the virialized region of each node are output if `outputVirialData` is set to true:

`nodeVirialRadius` The virial radius (following whatever definition of virial overdensity was selected in Galacticus) in units of Mpc;

`nodeVirialVelocity` The circular velocity at the virial radius (in km/s).

[3] Note that attempting to compute the velocity dispersion for a black hole or disk for example won't make any sense.

Part IV.

Development

In this Part we focus on how to modify Galacticus to meet your own needs. Galacticus is designed in a modular way to make it as simple as possible to introduce new implementations of physical processes or new galactic components without breaking the rest of the code. Nevertheless, some understanding of the structure of the code is necessary. In particular, Galacticus will happily compile and run calculations that make no physical sense whatsoever—it's up to you to ensure that the changes you make are physically reasonable and consistent with the behavior of the rest of the code.

14. Developing Galacticus

The following is a quickstart guide to making changes to the Galacticus source code and contributing them back to the project. Note that the preferred method to do this is through BitBucket.

14.1. Getting Started

It's easy to begin working with and changing the Galacticus source code. Assuming you have Mercurial ("`hg`") installed, just do:

```
hg clone https://abensonca@bitbucket.org/galacticusdev/galacticus galacticus
```

and you have a cloned copy of the Galacticus repository in the `galacticus` directory.

14.1.1. Using BitBucket

If you plan to contribute changes back to the Galacticus] project (please do!), you should consider using BitBucket. After you've created an account for yourself at BitBucket, you can "fork" the Galacticus repository to have your own working copy. This can be done as follows:

- visit the Galacticus repository on `BitBucket` at `https://bitbucket.org/abensonca/galacticus/overview`;
- click on the "Fork" button—you'll be presented with a form;
- fill out the form (setting a name for your fork, a short description, etc.), then click the "Fork repository" button;
- after the fork completes, you'll be taken to the overview page for your forked repository.

.

You'll now want to clone this forked repository to your local system. Click on the "Clone" button and copy the `hg clone` command presented there (you want the `SSH` version so that you can push changes back to this repository). Run this command on your local system to get a cloned copy of your new repository. You can now work with this repository, make any changes, and commit then. We'll discuss how to send these changes back to BitBucket, and back to the Galacticus project below.

14.2. Making Simple Changes

If you want to make some relatively minor changes to the Galacticus code, such as fixing a typo, adding a new filter, etc. you can just make changes directly on the `default` branch (i.e. at the point of active development). To do this, make sure you're at `default`:

```
hg pull
hg update default
```

Then make your changes, add new files, etc. Once you're done, first check if there have been any changes to `default` since you `pull`ed:

```
hg incoming
```

If any new changesets are shown, us `hg pull -u` to merge these in to your working copy. Then commit your changes:

```
hg commit
```

Your changes are now commited to your cloned repository.

14.2.1. Contributing Your Changes Back To Galacticus

Once you've committed your changes, you can contribute them back to the Galacticus project.

Via E-mail

If you just cloned the Galacticus repository directly you can send a patch containing your changes by e-mail to abenson@carnegiescience.edu. First, create the patch file using:

```
hg export -r begin:end > changes.diff
```

where `begin` and `end` are the first and last revisions that you want to include (you can specify more complicated sets of revisions of course). Then simply attached the `changes.diff` file to an e-mail. It will be merged into the Galacticus project using

```
hg import changes.diff
```

Using BitBucket

If you forked the Galacticus repository on BitBucket, you can now push your changes back to `BitBucket` using

```
hg push
```

If you want to contribute these changes back to the Galacticus project the best way to do so is to create a "pull request". Simply visit your forked respository on BitBucket and click the "Pull request" button. The form you're presented with allows you to choose which branch in your respository you want to send changes from, and which branch in the Galacticus project you want them contributed to. Add a title and description of your changes (and, optionally, check the "Close branch" box if you're done with this branch of development) then click "Create pull request". Assuming your code looks godo and works, it can then be pulled into the Galacticus project.

14.3. Making Bigger Changes

For bigger changes, particularly those where you're adding a new feature, we recommend using Mercurial's "feature branches". These provide a permanent record of for which feature each changeset was added. Using feature branches is straightforward. Begin with `default` and create a new branch:

```
hg update default
hg branch myNewFeature
```

where `myNewFeature` is a name for your feature branch. Then begin working, make changes, add new files etc. You can make commits when necessary (and it's good to make several small commits rather than one big one). You should merge `default` into your feature branch as often as possible to avoid them getting out of sync (which makes for difficulty later when you want to merge your feature branch back into `default`):

```
hg update myFeatureBranch
hg merge default
hg commit -m "merged default into myFeatureBranch"
```

Once the feature branch is stable, you can merge it back into `default`:

```
hg update default
hg merge myFeatureBranch
hg commit -m "merged myFeatureBranch"
```

Once you're done developing this feature, you should close the feature branch:

```
hg commit --close-branch -m "finished my feature"
```

Note that you can always go back and work on a feature branch later, after you have closed it. Just do:

```
hg up myFeatureBranch
hg merge default
hg commit -m "merged default into myFeatureBranch"
```

then continue to work with your feature branch as normal (don't forget to close it again when you're finished working with it).

14.4. Releases

Each release of Galacticus exists as a separate branch within the main Galacticus repository. To work with a particular release use

```
hg update v0.9.2
```

replacing the version number with whichever version you want. To get back to the development tip use

```
hg update default
```

14.4.1. Bug Fixes In Releases

To make a bugfix in a release, simply `hg update` to that release, fix the bug, and commit your changes. In many cases you'll want to fix the same bug in later releases and also in `default`. To do that, just `hg update` to each branch in turn, use `hg merge fixedBranch` (where "`fixedBranch`" is the name of the branch in which you fixed the branch, and then commit the merge. Once the bug has been fixed you can contribute the fix back to the Galacticus project using the methods described above.

14.5. The Galacticus Build System

Galacticus use a GNU Make-based build system. Dependencies are automatically discovered, and extensive meta-programming and preprocessing of source files is undertaken as part of the build. As a result, the build system is complicated. In this section the build system is described and detailed.

14.5.1. The Makefile

Files generated during the build (with the exception of final executables) are written to the directory specified by the `BUILDPATH` environment variable. This is normally `work/build/` (and is set by the `Makefile`) unless a special build configuration is requested.

14.5.2. Automatic Discovery

Interdependencies between source files, together with requirements for auto-generated code are automatically discovered during the build process by processing of source files. All such automatic discovery is described below.

Code Directive Parsing

The modular nature of Galacticus is enabled through the use of directives embedded within the source code (XML documents embedded as comment lines starting with a special tag) which instruct the build system how to link together the various functions. Parsing of these directives is handled by the `codeDirectivesParse.pl` script.

For `include` directives (which generate files that get included into the source), the script generates `$(BUILDPATH)/*.xml` files which describe the directive, and `$(BUILDPATH)/Makefile_Directives` which contains rules for building those include files.

The script also outputs a file `$(BUILDPATH)/directiveLocations.xml` which provides list of files that contain each directive. This is used by other scripts to permit rapid processing of files associated with each directive.

Allocatable Arrays

Types and ranks of allocatable arrays required by the Galacticus are discovered by the `allocatableArrays.pl` script. Source files are parsed for dimensionful, allocatable intrinsic types. All such combinations of type and rank are stored to a file `$(BUILDPATH)/allocatableArrays.xml`. The content of this file is later used to build appropriate memory allocation and deallocation functions (see §14.5.3).

Executable Files

Files which produce executables (including the `Galacticus.exe` executable) are discovered by the `scripts/build/findExecu` script, which generates rules describing how these files should be built, along with all of their dependencies, and writes them to `$(BUILDPATH)/Makefile_All_Execs`. All Fortran files in the `source` directory are parsed, and executable files are identified as those containing a `program` statement. All executables so identified are also added to the list of objects that will be built by `make all`.

Modules Provided

Determination of which files provide which Fortran modules is carried out by the `scripts/build/moduleDependencies.pl` script, which generates rules describing these dependencies (and how to build the module file by compiling the corresponding source file) and writes them to `$(BUILDPATH)/Makefile_Module_Dependencies`.

Additionally, rules are generated which describe how to build the following classes of file:

`*.mod.d` Contains a list of all object files upon which the module depends. Used in constructing the final set of objects which must be linked to build an executable.

`*.mod.gv` Contains a list of all source files upon which the module depends. Used in constructing GraphViz visualizations of dependencies.

`*.m` Contains a list of all modules provided by the corresponding object file. Used when building object files to test whether corresponding module files have been changed—allows avoidance of recompilation cascades if modules do not need to be updated (i.e. if the modules do not differ as judged by the `scripts/build/compareModules.pl` script).

Modules Used

Determination of which files use which Fortran modules is carried out by the `scripts/build/useDependencies.pl` script, which generates rules describing these dependencies and writes them to `$(BUILDPATH)/Makefile_Use_Dependencies`.

Additionally, rules are generated which describe how to build the following classes of file:

`*.d` Contains a list of all object files upon which each object file depends. Used in constructing the final set of objects which must be linked to build an executable.

`*.gv` Contains a list of all source files upon which each file depends by virtue of `use` statements. Used in constructing GraphViz visualizations of dependencies.

`*.fl` Contains a list of libraries upon which each object file depends. Used in constructing the final set of libraries which must be linked with the executable.

Included Files

Dependencies on files due to the use of "`include`" statements (in both Fortran and C source files) are discovered by the `scripts/build/includeDependencies.pl` script, which generates rules describing these dependencies and writes them to `$(BUILDPATH)/Makefile_Include_Dependencies`.

All Fortran and C/C++ source files in the `source` directory are parsed. Files specified in any include statement in a source file are added as a dependency of that source file if unless the source file is C/C++ and the named include file can be found in either `source/`, a standard system include path, or in any include path specified in the `GALACTICUS_CFLAGS` or `GALACTICUS_CPPFLAGS` environment variables.

Parameter Dependencies

All runtime parameters upon which a given executable may depend are discovered by the `scripts/build/parameterDependen` script, which generates XML files listing these parameters and writes them to `$(BUILDPATH)/<executableName>.parameters` All Fortran and C++ files in the `source` directory are parsed, and embedded parameter definitions (i.e. embedded XML blocks in lines beginning "!" for Fortran or "//" for C++ with root element `inputParameter`) are identified. Any files included into a source file are also searched. In the case of Fortran source files, `source/*.F90`, the corresponding preprocessed file, `$(BUILDPATH)/*.p.F90`, will be searched for parameter dependencies.

14.5.3. Code Generation

During build, code is generated automatically. Much of this is to link together functions within GALACTICUS (thereby permitting the modular nature of GALACTICUS), but also includes generation of various utility functions. All code generation is described below.

Memory Management Functions

Functions for allocation and deallocation of intrinsic-type arrays are generated by the `memoryManagementFunctions.pl` script. Utilizing the list of such types in the `$(BUILDPATH)/allocatableArrays.xml` file (see §14.5.2) overloaded functions are created for each type. These memory management functions keep track of the total amount of memory allocated.

`memoryManagementFunctions.pl`

Directives

Generation of code to implement the functionality of `include` directives is carried out by the `buildCode.pl` script. This script simply reads an XML file generated by the `codeDirectivesParse.pl` (see §14.5.2), calls the appropriate function to generate the necessary code, passes this through the preprocessor (see §15.2), and outputs the result to the appropriate include file.

Pre- and Post-processing

Before being compiled, all Fortran source files are passed through a tree-based preprocessor. The functionality of this preprocessor is described in §15.2. The preprocessor is invoked on each source file by the `preprocess.pl` script, which takes a `source/*.F90` file as input and outputs a preprocessed file `$(BUILDPATH)/*.p.F90`.

Output from the `gfortran` compiler is directed through the `postprocess.pl` script. This script currently performs the following functions:

- Builds a map from line numbers in the preprocessed source file back to line numbers in the unpreprocessed source, and translates line numbers in any error or warnings messages from the compiler into line numbers in the unpreprocessed source;
- Filters out incorrect warning messages about the lack of scalar finalizers (see GCC PR58175);
- Checks for unused function attributes in the source and filters out warnings about these unused functions.

14.5.4. Build Files

All files involved in the build process are summarized below.

`scripts/build/allocatableArrays.pl`: Discovers dimensionful, allocatable, intrinsic-typed arrays in the source code, and generates a summary of their types and ranks to the file `$(BUILDPATH)/allocatableArrays.xml` (see §14.5.2);

`scripts/build/buildCode.pl`: Acts upon `include` directives embedded in the source code and generates the corresponding include file (see §14.5.3);

`scripts/build/codeDirectivesParse.pl`: Discovers and parses directives embedded in source files. Generates `make` rules for the resulting dependencies, a file providing a mapping of which files contain each directive, and rule files describing how to build each include file resulting from a `include` directive;

`scripts/build/compareModuleFiles.pl`: Tests whether two Fortran module files are identical (ignoring timestamp information). Used to avoid updating modules when not necessary and so avoids recompilation cascades;

`scripts/build/includeDependencies.pl`: Discovers (and generates rules for) dependencies between files arising from the use of "`include`" statements (see §14.5.2);

`scripts/build/executableSize.pl`: Generates a file containing details of the size of each generated executable which can be used at runtime for memory reporting (see §14.5.2);

`scripts/build/findExecutables.pl`: Discovers (and generates rules for) files which generate executables (see §14.5.2);

`scripts/build/libraryDependencies.pl`: Determines the set of libraries that must be linked with each executable, and ensures they are ordered correctly for static linking;

`scripts/build/memoryManagementFunctions.pl`: Generates memory allocation/deallocation functions for allocatable intrinsic arrays (see §14.5.3);

`scripts/build/moduleDependencies.pl`: Discovers (and generates rules for) dependencies of module files on their source file statements (see §14.5.2);

`scripts/build/parameterDependencies.pl`: Determines all parameters upon which a given executable depends (see §14.5.2);

`scripts/build/postprocess.pl`: Postprocesses the output of the `gfortran` compiler to remove spurious warnings and provide line numbers in warning/error reports that point into the unpreprocessed source files (see §14.5.3);

`scripts/build/preprocess.pl`: Preprocesses Fortran source code to provide various extended functionality (see §14.5.3);

`scripts/build/useDependencies.pl`: Discovers (and generates rules for) dependencies between files originating from module `use` statements (see §14.5.2);

`$(BUILDPATH)/Makefile_All_Execs`: Contains rules describing how to build the executable file for each distinct program—generated by `scripts/build/findExecutables.pl` (see §14.5.2);

`$(BUILDPATH)/Makefile_Component_Includes`: Describes dependencies on the `nodeComponent` class hierarchy module on include files which implement specific functionality for individual node component implementations;

`$(BUILDPATH)/Makefile_Directives`: Describes rules for building include files for `include` directives;

`$(BUILDPATH)/Makefile_Include_Dependencies`: Contains rules describing the dependencies of source file on files included via "`include`" statements—generated by `scripts/build/includeDependencies.pl` (see §14.5.2);

`$(BUILDPATH)/Makefile_Module_Dependencies`: Contains rules which describe how to build module files from their source files (see §14.5.2);

`$(BUILDPATH)/Makefile_Use_Dependencies`: Contains rules which describe dependencies between source file originating from module `use` statements (see §14.5.2);

`$(BUILDPATH)/allocatableArrays.xml`: Contains a summary of the types and ranks of dimensionful, allocatable, intrinsic-type arrays for which memory allocation/deallocation functions will be generated (see §14.5.2);

`$(BUILDPATH)/directiveLocations.xml`: Contains a map of which files contain each code directive (see §14.5.2);

`$(BUILDPATH)/*.xml`: Contain rules for building files for `include` files (see §14.5.2);

`$(BUILDPATH)/*.d`: Contain lists of dependencies on object files, and are accumulated to find the final set of files which must be linked to build each executable (see §14.5.2 & §14.5.2);

`$(BUILDPATH)/*.m`: Contain lists of modules provided by each object file, and are used during testing whether modules have been updated (see §14.5.2);

`$(BUILDPATH)/*.fl`: Contain lists of libraries upon which each object file dependes, and are used to determine the final set of libraries which must be linked with each executable (see §14.5.2);

`$(BUILDPATH)/*.gv`: Contain lists of source files upon which each file depends and are used in the construction of GraphViz representations of file dependencies (see §14.5.2 & §14.5.2);

`$(BUILDPATH)/*.Inc`: Unpreprocessed include files generated from `include` dependencies (see §14.5.2);

`$(BUILDPATH)/*.inc`: Preprocessed include files generated from `include` dependencies (see §14.5.2);

`$(BUILDPATH)/*.p.F90`: Preprocessed Fortran source files (see §14.5.3);

`$(BUILDPATH)/*.parameters.xml`: Contain lists of parameters upon which an executable depends (see §14.5.2);

`$(BUILDPATH)/*.size`: Contain the size of each executable file built, and are used at runtime for memory reporting purposes—generated by `scripts/build/executableSize.pl` (see §14.5.2).

15. Coding Galacticus

15.1. Node Class Hierarchy Builder

The hierarchy of classes describing a merger tree node and its galaxy (i.e. `treeNode`, `nodeComponent`, `nodeComponentBasic`, `nodeComponentBasicStandard`, etc.) are built automatically to realize the set of component implementations and properties specified in source files.

15.1.1. Variable Definitions

Throughout the node objects builder code, Fortran variables are defined using a common specification. This is a simple hash containing the following entries:

intrinsic a string specifying the intrinsic Fortran type (`integer`, `logical`, `real`, `double precision`, `complex`, `double complex`, `type`, `class`, `procedure`);

type *[optional, except for* `type` *and* `class` *instrinics]* a string specifying the type of the variables;

attributes an array of strings specifying all attributes of the variables;

variables an array of strings giving the names of the variables.

For example, rank-1, long integer arguments which will not be modified in their function would be specified as:

```
{
  intrinsic  => "integer",
  type       => "kind_int8",
  attributes => [ "dimension(:)", "intent(in)" ],
  variables  => [ "argument1", "argument2" ]
}
```

15.1.2. The `$build` Data Structure

The `$build` data structure is used to accumulate all objects, variables, interfaces, and functions needed to build the class hierarchy used to represent nodes and components in Galacticus. At the end of the node objects build process, this object is processed to generate the required Fortran code. The following subsections describe how to add information to this object.

Component Classes and Implementations

A reference to a hash of structures which define all component classes, keyed by class name, is provided as `$build->{'componentClasses'}`. Each component class structure has the form:

```
{
 name    => <name of the class>,
 members => <reference to a hash of structures which define all member implementation of the class>
}
```

Each member implementation structure has the form:

```
{
 name       =>             <name of the implementation>,
 properties => property => <reference to a hash of structures which define all properties of this implem
}
```

Types

Definitions of derived types are accumulated to the hash `$build->{'types'}`, with the key of each entry being the derived type's name. Each derived type structure has the form:

```
{
 name           => <name of the derived type>
 comment        => <a description of the type>
 isPublic       => <1 if the class should have public visbility, 0 otherwise>
 dataContent    => <variable list containing all data for this class>
 boundFunctions =>
  [
   {
    type        => <procedure|generic>
    descriptor  => <a function descriptor data structure> [optional]
    name        => <name of the bound method>
    function    => <name of function to bind to> [optional; not required if descriptor is provided]
    description => <LaTeX-syntax description of the method> [optional; not required if descriptor is pro
    returnType  => <LaTeX-syntax return type of the method> [optional; not required if descriptor is pro
    arguments   => <LaTeX-syntax definition of all arguments to the method> [optional; not required if d
   }
  ]
}
```

The preferred approach is to provide a function descriptor (see §15.1.2), and to omit the `function`, `description`, `returnType`, and `arguments` entries (which will be inferred from the function descriptor). The `function`, `description`, `returnType`, and `arguments` entries will eventually be deprecated in favor of the `descriptor` entry.

Interfaces

Definitions of interfaces are accumulated to the hash `$build->{interfaces}`, with the key of each entry being the interfaces name. Each interface structure has the form:

```
{
 name           => <name of the interface>
 comment        => <a description of the interface>
 intrinsic      => <the intrinsic type of the function (or "void" for a subroutine)>
 data           => <variable list containing all arguments for this interface>
```

Functions

Definitions of functions are accumulated either to the list `$build->{'functions'}` or are included as the `descriptor` in a derived type structure (this is the prefered method for functions that *are* bound to a derived type; see §15.1.2). Each function structure has the form:

```
{
 type        => <the type of function>
 name        => <function name>
 description => <LaTeX-syntax description of the function>
 modules     => <list of names of modules required by the function> [optional]
 variables   => <list of variables required by the function> [optional]
 content     => <the code of the function (excluding opener, closer, and variable definitions)>
}
```

Module-scope Variables

Any module-scope variables can be pushed to the array `{$build->{'variables'}}`, using the usual definition format described in §15.1.1.

15.2. Galacticus Preprocessor Directives

Galacticus has its own preprocessor for Fortran source files. This preprocesses parses each source file into an internal tree representation, performs various manipulations on that tree, and then outputs the preprocessed file for compilation. The preprocessor is used to automate and standardize many common tasks, through the inclusion of directives into the source code. Directives are specified in comment lines beginning `!#`, and are written in XML. The remainder of this section describes the various preprocessor functionalities, and gives examples of their usage.

15.2.1. Source Code Introspection

The source code introspection functionality allows automated generation of information about the source code. Specifically:

- The directive `{introspection:location}` in source code will be replaced with a character string giving a backtrace of the current location in the code, including any function, module, and file (including line number).

15.2.2. Function Attributes

Galacticus allows the specification of attributes for functions which alter the way the compiler treats them. Function attributes are specified as:

```
!$GLC function attributes {attributes} :: {functionNames}
```

where `{attributes}` is a space-separated list of attributes, and `{functionNames}` is a space-separated list of functions to apply these functions to. Function attribute directives may appear anywhere in a file containing the named function, but it is good practice to locate them immediately before the function.

Currently, the supported attributes are:

`unused` The function is marked as being *possibly* unused. Compiler warnings about unused functions will be suppressed for this function.

15.2.3. Source Digest

The `sourceDigest` directive will generate an MD5 hash hash of the source code of the file in which the directive is placed, along with the source code of any files upon which it depends. This can be useful in

generating unique labels (e.g. to use as suffixes in file names) which automatically update if the source code is modified. To generate a source digest simply use:

```
!# <sourceDigest name="mySourceDigest"/>
```

A source digest will be generated and stored as a `character(len=22)` variable called `mySourceDigest`.

15.2.4. Object Builder

When constructing instances of a class from a provided parameter set, a common pattern is to need to construct other objects based on those parameters, which will be used by the instance. For example, a transfer function class might require a cosmological parameters object for its operation. In such cases, we often want to use the default instance of the required class unless a different instance is explicitly specified. The `objectBuilder` directive automates this process.

As an example, the following constructor requires an instance of the `cosmologyParameters` class, which is passes to an internal constructor:

```
function myConstructorParameters(parameters)
  use Input_Parameters2
  implicit none
  type(myClass                    )                :: myConstructorParameters
  type(inputParameters            ), intent(in    ) :: parameters
  class(cosmologyParametersClass), pointer         :: cosmologyParameters_

  !# <objectBuilder class="cosmologyParameters" name="cosmologyParameters_" source="para
  myConstructorParameters=myConstructorInternal(cosmologyParameters_)
  return
end function myConstructorParameters
```

The `objectBuilder` directive will assign a member of the class specified by the `class` attribute to the variable specified by the `name` attribute. If the parameter set specified by the `parameters` attribute contains an explicit definition of the relevant class, that definition will be used to construct the instance. Otherwise, the parent parameter set will be checked for a definition of the relevant class, and so on. If no definition is found when the root parameter set is found the default instance will be used. Note that objects created by an `objectBuilder` directive are directly associated with the element in the input parameters XML document from which they were created. Therefore, if a later `objectBuilder` requires the object from that same XML element, it will reuse the one previously created.

The `objectBuilder` directive by default searches for a parameter named `{class}Method` where `{class}` is the value of the `class` attribute in the directive. An alternative name may be specified via the addition of a `parameterName` attribute.

In cases where an explicit `parameterName` attribute is given, it is possible to specify an explicit default for the object if no such named parameter is found. (Where no explicit `parameterName` attribute is given the global default of the class is used.) A default is specified by adding a `default` element to the directive, which should contain default specification for the object (and any subobjects) in the usual format. For example:

```
!# <objectBuilder class="massDistribution" parameterName="diskMassDistribution" name="diskMassDistribut
!#  <default>
!#   <diskMassDistribution value="exponentialDisk">
!#    <dimensionless value="true"/>
!#   </diskMassDistribution>
!#  </default>
!# </objectBuilder>
```

15.2.5. Object Destructor

This directive can (and should) be used to destroy objects built by the `objectBuilder` directive. The `objectDestructor` directive automates the process of deciding if these objects should be destroyed (or merely have pointers to them nullified). As an example, the following destructor destroys two associated objects:

```
 subroutine simpleDestructor(self)
   implicit none
   type(powerSpectrumPrimordialTransferredSimple), intent(inout) :: self

   !# <objectDestructor name="self%transferFunction_"         />
   !# <objectDestructor name="self%powerSpectrumPrimordial_"/>
  return
 end subroutine simpleDestructor
```

15.2.6. Constructor Assignments

A common requirement in object constructors is to assign the values of arguments to the constructor to corresponding entries in the object. The `constructorAssign` directive performs this assignment for a list of comma-separated variables. For example:

```
 function stellarMassConstructorInternal(massThreshold)
   implicit none
   type             (galacticFilterStellarMass)                 :: stellarMassConstructorIn
   double precision                           , intent(in    ) :: massThreshold
   !# <constructorAssign variables="massThreshold"/>
   return
 end function stellarMassConstructorInternal
```

will cause the value of the `massThreshold` argument to `stellarMassConstructorInternal%massThreshold`. If an argument name is prefixed with `*` in the variables list, pointer assignment is used instead of standard assignment.

15.2.7. State Storing

Galacticus supports storing its internal state to file to allow restarts (see §2.3.2). Code to store and restore the internal state of objects of a given class can be generated automatically through use of the `stateStorable` directive. An example is:

```
!# <stateStorable class="table">
!#  <table1DGeneric>
!#   <restoreTo variables="reset" state=".true."/>
!#   <exclude variables="staticData"/>
!#  </table1DGeneric>
!#  <table2DLinLinLin>
!#   <restoreTo variables="resetX, resetY" state=".true."/>
!#  </table2DLinLinLin>
!# </stateStorable>
```

This specifies that the `table` class (and all child classes) can and should be stored to file as part of the representation of the internal state. Code to store and restore all data associated with any object of this

class (as well as restoring polymorphic objects to the correct type) will be generated. If certain variables of the classs or subclass should be restored to specific values this can be specified through a `restoreTo` element placed within an element with the name of the class or subclass (e.g. the `table1DGeneric` in the above example). The `restoreTo` element should specify a comma-separated list of one or more variables to set in its `variables` attribute, and the state to which they should be restored in its `state` attribute. Any variables which should be excluded from state store/restore (e.g. if their values are known to be determined statically at construction) can be specified via a `exclude` element—a list of variables to exclude should be given as a comma-separated list in its `variables` attribute.

15.2.8. Conditional Call

In some instances it is useful to be able to call a function with different combinations of optional arguments depending on certain conditions. (For example, if some combinations of optional arguments are mutually exclusive.) This can be achieved using the `conditionalCall` directive. An example is:

```
!# <conditionalCall>
!#  <call>self=massDistributionBetaProfile(beta{conditions})</call>
!#  <argument name="densityNormalization" value="densityNormalization" parameterPresent="parameters"/>
!#  <argument name="mass"                 value="mass"                 parameterPresent="parameters"/>
!#  <argument name="outerRadius"          value="outerRadius"          parameterPresent="parameters"/>
!#  <argument name="coreRadius"           value="coreRadius"           parameterPresent="parameters"/>
!#  <argument name="dimensionless"        value="dimensionless"        parameterPresent="parameters"/>
!# </conditionalCall>
!# <inputParametersValidate source="parameters"/>
```

The `call` element specifies the function call, and contains the special sequence `{conditions}` which will be replaced with the conditionally-present arguments. One or more `argument` elements should specify the various arguments which should be included in the call. For each such element the `name` attribute specifies the name of the dummy argument in the called function, the `value` attribute specifies the value (or variables) to pass this this dummy argument. A condition for inclusion of the argument must also be specified. In the above, the special `parameterPresent` condition is used. The argument will be included in the call if a parameter with a name matching the `name` attribute exists in the parameter set named in the `parameterPresent` attribute. Alternatively a `condition` attribute can be given. An argument is included in the call if the expression given in the `condition` attribute evaluates to true.

Code will be generated to call the function with all possible combinations of arguments.

15.2.9. Optional Arguments

Fortran supports optional arguments to functions, but does not provide for a default value if those arguments are not present. The `optionalArgument` directive allows a default value to be specified. In the following example, a default value is defined for the `units` argument:

```
double precision function simpleHubbleConstant(self,units)
  implicit none
  class  (cosmologyParametersSimple), intent(inout)           :: self
  integer                           , intent(in   ), optional :: units
  !# <optionalArgument name="units" defaultsTo="hubbleUnitsStandard" />

  select case (units_)
  case (hubbleUnitsStandard)
     ! Return the value using the default units.
```

```
    case ....
       ! Return the value using some other units.
    end select
    return
  end function simpleHubbleConstant
```

The `optionalArgument` directive should appear after variable declarations and before any attempt to use the optional argument, and should have two attributes, `name` and `defaultsTo` which give the name of the argument variable and its default value respectively. The preprocessor will add a new variable with the same name plus an underscore suffix, and will ensure that it is initialized to the default value if the optional variable is not present, otherwise setting it to the value of the optional variable.

Note that for optional arguments that are `intent(out)` or `intent(inout)` the preprocessor currently *does not* ensure that the value of the new variable is copied back to the argument prior to exit from the function.

15.2.10. Enumerations

The `enumeration` directive allows specification of an enumeration (a set of labels), and (optionally) functions to decode such a label from user input. The following example illustrates this usage:

```
module Cosmology_Parameters

  !# <enumeration>
  !#  <name>hubbleUnits</name>
  !#  <description>Specifies the units for the Hubble constant.</description>
  !#  <visibility>public</visbility>
  !#  <validator>yes</validator>
  !#  <encodeFunction>yes</encodeFunction>
  !#  <entry label="standard" />
  !#  <entry label="time"     />
  !#  <entry label="littleH"  />
  !# </enumeration>

contains

subroutine Test_Enumeration()
    use ISO_Varying_String
    implicit none
    class(cosmologyParametersClass), pointer :: cosmologyParameters_
    cosmologyParameters_ => cosmologyParameters()
    write (0,*) "Enumeration contains ",hubbleUnitsCount," entries from ",hubbleUnitsMin,"
    write (0,*) "Hubble constant in little-h units is: ",cosmologyParameters_%HubbleConstan
    if (enumerationHubbleUnitsEncode('hubbleUnitsStandard') == hubbleUnitsStandard) then
      write (0,*) "Enumeration decoding succeeded"
    else
      write (0,*) "Enumeration decoding failed"
    end if
    if (enumerationHubbleUnitsEncode('standard',includesPrefix=.false.) == hubbleUnitsStand
      write (0,*) "Enumeration decoding succeeded"
    else
      write (0,*) "Enumeration decoding failed"
```

```
    end if
    write (0,*) "Name from time value is '",char(enumerationHubbleUnitsDecode(hubbleUnits
  return
end subroutine Test_Enumeration

end module Cosmology_Parameters
```

Enumerations must be defined in the declaration section of a `module`. The encoding and decoding functions will only be generated if the `encode` element is present and has content `yes`. The enumeration variables are given `public` visibility by default—this can be overridden using the `visibility` element. If the `validator` element is present and set to `yes` then a function is created which will return true if the given value is a valid one for the enumeration. Additionally, if the `validator` element is present and set to `yes` variables are created giving a count of the number of entries in the enumeration, along with the minimum and maximum values in the enumeration. Note that the `description` element is used to generate an entry for the enumeration in the document, and so should be written in LaTeX syntax.

15.2.11. Input Parameters

The `inputParameter` directive reads an input parameter and assigns the appropriate value to the given variable. `inputParameter` directives must occur within the main body of a function, subroutine, or program. A default value can be specified if desired. The following example illustrates this usage:

```
subroutine simpleParametersRead()
  implicit none
  double precision :: hubbleConstant
  !# <inputParameter>
  !#   <name>HubbleConstant</name>
  !#   <source>myParameters</source>
  !#   <variable>hubbleConstant</variable>
  !#   <defaultValue>69.7d0</defaultValue>
  !#   <defaultSource>(\citealt{hinshaw_nine-year_2012}; CMB$+H_0+$BAO)</defaultSource>
  !#   <description>The present day value of the Hubble parameter in units of km/s/Mpc.</
  !#   <type>real</type>
  !#   <cardinality>0..1</cardinality>
  !# </inputParameter>
  if (hubbleConstant < 0.0d0) write (0,*) "The universe is collapsing!"
  return
end subroutine simpleParametersRead
```

In this case, the value of the `HubbleConstant` parameter is assigned to the `hubbleConstant` variable, with a default of 69.7 if no value was specified in the input parameter file. If the `source` element is present, the parameter will be read from the named `inputParameters` set, otherwise the parameter will be read from the top-level of the parameters file. The `defaultSource`, `<description>The`, `type`, and `cardinality` elements are used only for adding an entry for the input parameter to the documentation, and so should be written in LaTeX syntax.

It is also possible to specify a set of parameter which iterate over names defined by other directives. The following example would read one parameter named "`fileNameForXXXXXXIMF`" where "`XXXXXX`" equals the `name` element each `imfRegisterName` dircetive:

```
!# <inputParameter>
!#   <iterator>fileNameFor(#imfRegisterName->name)IMF</iterator>
!#   <source>parameters</source>
```

```
!#    <variable>fileNames(IMF_Index("$1"))</variable>
!#    <defaultValue>Galacticus_Input_Path()//"data/SSP_Spectra_imf$1.hdf5"</defaultValue
!#    <description>The name of the file of stellar populations to use for the named \gls
!#    <type>string</type>
!#    <cardinality>0..1</cardinality>
!# </inputParameter>
```

15.2.12. Input Parameter Lists

The `inputParameterList` directive will construct a `varying_string` array containing the names of all input parameters which are defined in the unit in which the directive appears. The name of the array is specified by the `label` attribute of the `inputParameterList` directive. If a `source` attribute is specified in the directive then only parameters being read from the named variable will be included in the list, otherwise any parameters read will be included. Such a list can be used to validate the names of parameters passed to a function for example.

15.2.13. Function Classes

Most[1] of the internal functionality within Galacticus is provided by "function classes". These are classes (in the object oriented sense) which model some particular physical entity or concept (e.g. the underlying cosmological model) and provide one or more functions associated with that entity or concept. A default implementation of each function class can be selected at run-time, allowing for simple user-defined control of model behavior. A function class is specified by a `functionClass` directive, together with one or more implementations of the function class specified by their own directives.

An example of a function class directive, which defines a class for cosmological parameters (which in this case, for simplicity, consists of just the Hubble constant) is given below:

```
!# <functionClass>
!#  <name>cosmologyParameters</name>
!#  <descriptiveName>Cosmological Parameters</descriptiveName>
!#  <description>Object providing various cosmological parameters.</description>
!#  <default>simple</default>
!#  <defaultThreadPrivate>no</defaultThreadPrivate>
!#  <stateful>no</stateful>
!#  <calculationReset>no</calculationReset>
!#  <method name="HubbleConstant" >
!#   <description>Return the Hubble constant at the present day. The optional {\normalfo
!#   <type>double precision</type>
!#   <pass>yes</pass>
!#   <argument>integer, intent(in    ), optional :: units</argument>
!#  </method>
!# </functionClass>
```

The directive should contain the following elements:

`name` The name of this function class.

`descriptiveName` A descriptive name for the function class, suitable for inclusion in the documentation.

`description` A description of the purpose of this function class. This description will be included into the documentation so should be written in LaTeX syntax.

[1] At this time the Galacticus code base is being transitioned to use this approach.

`default` The default implementation to use for this function class if no choice is made in the input parameter file.

`defaultThreadPrivate` *(optional)* If present and set to `yes` then the default implementation of this function class will be made OpenMP thread private. Otherwise, the default implementation is shared between threads. A thread private default implementation can be useful if the function class may need to generate look-up tables unique to each thread on the fly for example.

`calculationReset` *(optional)* If present and set to `yes` then the default implementation of the function class is assumed to possibly want to reset its calculations when the active node changes (see §16.4.3). In this case, an additional method is generated for the function class: `calculationReset` with interface:

```
subroutine calculationReset(self,thisNode)
  class  (functionClassBaseName), intent(inout)           :: self
  type   (treeNode             ), intent(inout), pointer :: thisNode
end subroutine calculationReset
```

This method has a null implementation for the base class of the function class, but can be overridden to reset calculations of any given implementation.

`method` Each `method` element defines a method which the function class will support, the name of which is given by a `name` attribute. The method definition must contain the following elements:

`description` A description of this method (in LaTeX syntax) suitable for inclusion into the documentation.

`type` The type of the function (e.g. `double precision`; use `void` for a subroutine).

`pass` If `yes` pass the object that the method was called on as the first argument.

`argument` *(optional)* Zero or more declarations (in standard Fortran syntax) for the method arguments (there is no need to specify the declaration for the object upon which the method was called in the case where this object is passed to the method function).

Implementations of this function class must be declared with a directive having the same name as the function class (i.e. `cosmologyParameters` in the example above). The directive must give the name of the implementation as an attribute, and a description of the implementation suitable for inclusion into the documentation. Each implementation should be placed in a separate file—the preprocessor will find these files and merge the implementations and function class definition into a single file for compilation. Each implementation should declare a class which extends the basic function class, constructor interfaces, and any module-scope data required by the class. The implementation should also define all necessary functions required by the class (separated from the declarations by a `contains` keyword). An example is given below:

```
!# <cosmologyParameters name="cosmologyParametersSimple">
!#  <description>Provides the Hubble constant: $H_0$.</description>
!# </cosmologyParameters>
type, extends(cosmologyParametersClass) :: cosmologyParametersSimple
   private
   double precision :: HubbleConstantValue
 contains
   final     ::                    simpleDestructor
   procedure :: HubbleConstant  => simpleHubbleConstant
end type cosmologyParametersSimple
```

```
  interface cosmologyParametersSimple
     module procedure simpleDefaultConstructor
     module procedure simpleConstructor
  end interface cosmologyParametersSimple

contains

  function simpleDefaultConstructor()
    implicit none
    type(cosmologyParametersSimple) :: simpleDefaultConstructor

    ! Construct an instance of this class using a value of the Hubble constant read from th
    !# <inputParameter>
    !#   <name>H_0</name>
    !#   <variable>simpleDefaultConstructor%HubbleConstantValue</variable>
    !#   <defaultValue>69.7d0</defaultValue>
    !#   <defaultSource>(\citealt{hinshaw_nine-year_2012}; CMB$+H_0+$BAO)</defaultSource>
    !#   <description>The present day value of the Hubble parameter in units of km/s/Mpc.</
    !#   <type>real</type>
    !#   <cardinality>0..1</cardinality>
    !# </inputParameter>
    return
  end function simpleDefaultConstructor

  function simpleConstructor(HubbleConstant)
    implicit none
    type            (cosmologyParametersSimple)                :: simpleConstructor
    double precision                           , intent(in    ) :: HubbleConstant

    ! Construct an instance of this class using a value of the Hubble constant provided di
    simpleConstructor%HubbleConstantValue=HubbleConstant
    return
  end function simpleConstructor

  elemental subroutine simpleDestructor(self)
    implicit none
    type(cosmologyParametersSimple), intent(inout) :: self

    ! Do any clean-up required by this class when an instance goes out-of-scope.
    return
  end subroutine simpleDestructor

  double precision function simpleHubbleConstant(self,units)
    implicit none
    class  (cosmologyParametersSimple), intent(inout)           :: self
    integer                           , intent(in    ), optional :: units

    ! Do whatever is necessary to return the Hubble constant in the appropriate units.
```

```
    return
  end function simpleHubbleConstant
```

In the above example, we define a "simple" implementation of the cosmologyParameters class. Key points are:

Name: The name should always be prefixed with the function class name. In this case, we have a `simple` implementation of the `cosmologyParameters` function class, and so our name is `cosmologyParametersSimple`.

Extends: The base class for the function class is always the function class name suffixed with `Class`, in this case `cosmologyParametersClass`. Implementations must always be extensions of either this base class, or of another implementation.

Procedures: The implementation must define procedures for all methods of the function class, *except* for where a method specified a `code` element in the function class directive (in which case a procedure for the method may still be optionally defined). Specification of a `final` function is encouraged.

Constructors: The implementation must specify at least one constructor, which takes no arguments (usually known as the default constructor). This constructor must create an instance of the implementation, setting any parameters from the input parameter file as necessary. Additional constructors may be defined as required.

Procedures: All required procedures (including constructors and destructors) should be given after a line containing the `contains` keyword. GALACTICUS coding policy is that all procedures associated with an implementation should be prefixed with the implementation name, `simple` in this case.

Functionality to store and restore the state (see §2.3.2) of classes built via a `functionClass` directive are automatically built. If variables of a given implementation should be restored to a specific state, this can be specified by adding a `restoreTo` element to the directive declaring the implementation. The `restoreTo` element should specify a comma-separated list of one or more variables to set in its `variables` attribute, and the state to which they should be restored in its `state` attribute. Any variables which should be excluded from state store/restore (e.g. if their values are known to be determined statically at construction) can be specified via a `exclude` element—a list of variables to exclude should be given as a comma-separated list in its `variables` attribute.

15.2.14. Generic Programming

The preprocessor supports generic programming by allowing generic types to be defined, which are automatically expanded to a set of specific types. Significant flexibility is provided to allow control over how each specific type is handled. A generic type is specific via a `generic` directive such as:

```
  !# <generic identifier="Type">
  !#  <instance label="Logical"          intrinsic="logical"
outputConverter="regEx|(.*)|char($1)|"/>
  !#  <instance label="Integer"          intrinsic="integer"
outputConverter="regEx|(.*)|$1|"       />
  !#  <instance label="Double"           intrinsic="double precision"
outputConverter="regEx|(.*)|$1|"       />
  !#  <instance label="LogicalRank1"     intrinsic="logical          , dimension(:)" outputC
  !#  <instance label="IntegerRank1"     intrinsic="integer          , dimension(:)" outputC
/>
  !#  <instance label="DoubleRank1"      intrinsic="double precision , dimension(:)" outputC
/>
  !# </generic>
```

The above defines a generic type, which will be identified using the label "`Type`". The directive contains several `instance` elements, each of which specifies a specific type which should be implemented for the generic type. Each instance can contain an arbitrary number of attributes which specify strings or regular expressions which will be used to construct the specific implementation.

A generic directive applies to the entire unit within which it is scoped. The preprocessor will examine every element within that unit. If a generic tag (see below) is found in the opening of any subunit, that entire subunit is copied once for each instance, and any generic tags replaced with the appropriate content from the `instance` element. Where a generic tag is found in a non-opening line (and that line is not contained within a subunit whose opener *does* contain a generci tag), the line itself is replicated in the same way.

An example of the usage of generic tags using the above generic directive is:

```
type :: exampleType
   private
   .
   .
   .
 contains
   final     ::         exampleTypeDestroy
   procedure ::         exampleTypeSet{Type|label}
   generic   :: set => exampleTypeSet{Type|label}
end type exampleType

contains

subroutine exampleTypeSet{Type|label}(self,setValue)
  implicit none
  class           (exampleType), intent(in   ) :: self
  {Type|intrinsic}             , intent(in   ) :: setValue

  {Type|match|^Logical|! Do something to set a logical value.|! Do something different to s
  write (0,*) "Value is: ",{Type|outputConverter|setValue}
end subroutine exampleTypeSet{Type|label}
```

In this example, the `exampleType` class is defined to have a generic `set` method. The presence of the `{Type@|@label}` generic tag will cause those lines to be replicated with the tag replaced by the content of the `label` attribute of each instance of the generic type. In the contained subroutine, a generic tag appears in the opener. As such, the entire subroutine will be replicated once for each instance of the generic type, and the generic tags replaced as appropriate.

When a generic instance attribute begins with `regEx`, matching generic tags are handled differently. In particular, a match-and-replace regular expression is applied to the third element of the generic tag (elements are separated by broken vertical bar characters). The match and replace components of the regular expression are defined in the instance attribute, once again separated by broken vertical bars.

Finally, the special generic tag `match` acts as a ternary operator. If the regular expression specified in the third element matches the `label` attribute of a specific instance, the generic tag is replaced with its fourth element, otherwise it is replaced with its fifth element.

15.3. Numerical Tools

Galacticus provides a variety of tools to solve basic numerical problems. These can be found in files `source/numerical.*`. Galacticus makes use of the GNU Scientific Library for many of these tools, but typically provides a higher-level wrapper around those functions, providing a cleaner interface and, in some cases, additional functionality.

15.3.1. Finding Roots of Equations

Tools for solving equations of the form $f(x) = 0$ are provided by the `rootFinder` object (available via the `Root_Finder` module). Typical use of this object is as follows:

```
! Import the module.
use Root_Finder
...
! Create a rootFinder object - make it OpenMP threadprivate so it can be used
! simultaneously by all threads.
type(rootFinder), save :: finder
!$omp threadprivate(finder)
...
! Check if our root finder has been initialized.
if (.not.finder%isInitialized()) then
  ! Specify the function that evaluates f(x).
  call finder%rootFunction    (myRootFunction                          )
  ! Specify the type of root-finding algorithm - this is optional (Brent's
  ! method will be used by default).
  call finder%type            (FGSL_Root_fSolver_Brent                 )
  ! Specify the tolerances to use in finding the root. Both arguments are
  !optional - values of 1.0d-10 will be used for both absolute and relative
  ! tolerance by default.
  call finder%tolerance       (toleranceAbsolute,toleranceRelative)
  ! Specify how the initially provided range can be expanded to bracket the
  ! root. This is optional - if not provided no range expansion will be attempted.
  call finder%rangeExpand                                              &
       & (                                                             &
       &  rangeExpandDownward          =0.5d0                        , &
       &  rangeExpandUpward            =2.0d0                        , &
       &  rangeExpandType              =rangeExpandMultiplicative    , &
       &  rangeDownwardLimit           =1.0d-3                       , &
       &  rangeUpwardLimit             =1.0d+3                       , &
       &  rangeExpandDownwardSignExpect=rangeExpandSignExpectPositive, &
       &  rangeExpandUpwardSignExpect  =rangeExpandSignExpectNegative
       & )
end if
x=finder%find(rootGuess=1.0d0)
.
.
.
double precision function myRootFunction(x)
  implicit none
```

```
   double precision, intent(in   ) :: x
   ...
   return
end function myRootFunction
```

The above example begins by importing the `Root_Finder` module and then creating a `rootFinder` object called `finder`. This is made OpenMP `threadprivate` so that it may be used simultaneously by all threads. The first step is to initialize `finder`—the `isInitialized` method tells us if this has already happened. The most important step is to specify the function that will evaluate $f(x)$. This is done via the `rootFunction` method—once done, the `rootFinder` object is marked as initialized (and the `isInitialized` method will return `true`). All other initialization steps are optional. In this example, we use the `type` method to specify that the `Brent` algorithm should be used for root finding. Any valid GSL-supported root finding algorithm can be used. We then use the `tolerance` method to specify both the absolute and relative tolerances in the x variable that must be attained to declare the root to be found. Both arguments are optional—default values of 10^{-10} will be used if either tolerance is not specified.

The final step of initialization is to call the `rangeExpand` method. This specifies how the initial guessed value or range for x should be expanded to bracket the root. If you plan to always specify an initial range, and know that it will always bracket the root, you do not need to specify how the range should be expanded. In this case we've specified that range expansion is multiplicative—that is, the lower and upper values of x defining the range will be multiplied by fixed factors until the root is bracketed—via the `rangeExpandType=rangeExpandMultiplicative` option. Alternatively, additive expansion is possible using `rangeExpandType=rangeExpandAdditive`. The factors by which to multiply the lower and upper bounds of the range (or the factor to add in the case of additive expansion) are specified by the `rangeExpandDownward` and `rangeExpandUpward` options. Iit is possible to specify absolute lower/upper limits to the range via the `rangeDownwardLimit` and `rangeUpwardLimit` options. The range will not be expanded beyond these limits—if the root cannot be bracketed without exceeding these limits an error condition will occur. Finally, it is possible to indicate the expected sign of $f(x)$ at the lower and/or upper limits via the `rangeExpandDownwardSignExpect` and `rangeExpandUpwardSignExpect` options. Valid settings are `rangeExpandSignExpectNegative`, `rangeExpandSignExpectPositive`, and `rangeExpandSignExpectNone` (the default—implying that there is no expectation for the sign). If the sign of $f(x)$ is specified, then range expansion will stop once the expected sign is found. This can often improve efficiency, by allowing the range expander to expand the range in only one direction, resulting in a narrower range in which to search for the root.

Finally, we use the `find` method to return the value of the root. The first argument to `find` is the name of the function that evaluates $f(x)$. Additionally, we must supply either `rootGuess` (a scalar value guess to use as the initial value for both the lower and upper values of the range—note that range expansion must be allowed in this case), or `rootRange` (a two-element array to use as the initial lower and upper values of the range bracketing the root).

The function evaluating $f(x)$ must have a form compatible with that shown for `myRootFunction` in the above example.

15.4. Computation Dependencies and Data Files

In many situations, some module in Galacticus might want to perform a calculation and then store the results to a file so that they can be reused later. A good example is the `CAMB` transfer function method (see §12.6.5), which computes a transfer function using CAMB and stores this function in a file so that it can be re-read next time, avoiding the need to recompute the transfer function. A problem arises in such cases as the calculation may depend on the values of parameters (in our example, the transfer function will depend on cosmological parameters for example). We would like to record which parameter values this calculation refers to, perhaps encoding these into the file name, so that we can reuse these data in a

future run only if the parameter values are unchanged. Given the modular nature of Galacticus it is impossible to know in advance which parameters will be relevant (e.g. does the cosmological parameter implementation have a parameter that describes a time varying equation of state for dark energy?).

To addres this problem, Galacticus provides a mechanism to generate a unique descriptor for a given object. This descriptor encodes the parameter used to construct the object, and recursively includes the parameters used to construct any other object which is composited. A long-form (human readable) descriptor is returned by the `descriptor` method associated with all `functionClass` objects. Additionally, the `hashedDescriptor` method will return an MD5 hash of the descriptor, which will be unique (up to collisions) and can be used to idetify the object both internally and, for example, when used as a suffix to file names. If the optional `includeSourceDigest` argument is set to true in the `hashedDescriptor` method then the hashed descriptor will include a hash of the source code of the object (and all composited objects) such that the descriptor will change should the source code be changed.

15.5. Optimization

In designing Galacticus, we opted for simplicity and clarity over speed. However, there are numerous parts of the code where optimization has been performed without a significant loss of clarity. In this section we discuss some of the techniques used.

15.5.1. Unique IDs and Stored Properties

Frequently, a given property of a node may be required in many different aspects of the calculation. For example, the dark matter halo virial radius is used extensively in several distinct calculations within Galacticus. Frequently such calculations are performed for the same node, with the same properties several times[2]. Obviously this is inefficient. It can be advantageous in such cases to store the result of a calculation and, if the function is called again with the same unchanged node to simply return the stored value. Galacticus facilitates this by two features.

The first feature is the "unique ID"—an integer number assigned to each node in Galacticus and which uniquely identifies a node (i.e. no two nodes processed in a Galacticus run will have the same unique ID). This number, which can be retrieved using the `uniqueID` property of a tree node, can be recorded each time a function is called. If called again for a node with the same unique ID as the previous call, the function can simply return the same answer as on the previous call.

The second feature accounts for the fact that the properties of a node will change, so even if a function is called on a node with the same unique ID it may occasionally need to recompute its result. Galacticus provides a calculation reset task (see §16.4.3). All such tasks are performed just prior to the computation of derivatives for a node being evolved. A function can register a calculation reset task and use it to flag that it must update its calculations even if called again with the same node.

15.6. Global Functions

In very exceptional circumstances it is necessary to subvert the module hierarchy used by Galacticus to permit one module to call a function in a higher level module[3]. Examples of where this approach is necessary usually involve initial bootstrapping (i.e. to establish halo density contrasts, which requires knowledge of the halo density profile, which in turn requires knowledge of the halo density contrast...).

[2] For example, Galacticus's ODE solver will fix the properties of a node and then request that derivatives of all properties be computed. Some functions will then be called multiple times for the same node with unchanged properties.

[3] This usually arises because circular dependencies would arise if the called function were placed in a lower level module.

Galacticus provides for global functions which facilitate this—specifically, it is possible to generate function pointers to higher level functions which are accessible via a very low-level module. To create a globally callable copy of a function use add the follow prior to the function definition:

```
!# <functionGlobal>
!#  <unitName>myFunction</unitName>
!#  <type>double precision</type>
!#  <arguments>double precision , intent(in   ) :: mass, time</arguments>
!# </functionGlobal>
```

Here, `myFunction` is the name of the function to make global, while the `type` and `arguments` (of which there may be more than one) elements are used to generate a suitable interface for the function. At run-time, a pointer to this function is then available from the `Functions_Global` module, named `myFunction_`. Note that `Functions_Global_Set` provided by the `Functions_Global_Utilities` module must be called once to initialize these global function pointers prior to their use.

15.7. Galacticus Metadata

Galacticus can collect metadata on its own activity. This is useful for profiling the code for example.

15.7.1. OpenMP Critical Section Wait Times

Galacticus makes use of OpenMP for parallel operation. OpenMP `critical` sections are used throughout to limit access to parts of the code which must be executed in serial. Threads will block at each `critical` section if another thread is currently within it. Galacticus can profile the amount of time spent waiting at each named OpenMP `critical` section across all threads. To enable this profiling Galacticus should be compiled with `-DOMPPROFILE` added to the compile options (in `GALACTICUS_FCFLAGS`). This will cause profiling instructions to be added to the source code prior to compilation. When this instrumented executable is run an `metaData/openMP/` group will be added to the output file. This group contains two datasets, `criticalSectionNames` and `criticalSectionWaitTimes`. The first lists the names of all named OpenMP `critical` sections, while the second lists the total number of seconds (across all threads) spent waiting at each section. A script is provided to analyze this metadata:

```
./scripts/aux/openMPCriticalWaitProfile.pl <modelFileName>
```

This script will analyze the OpenMP `critical` section wait time metadata in the named file, reporting the total time spent waiting at critical sections followed by a rank ordered list of the top ten sections by wait time. This can be useful for assessing whether optimization might help to reduce OpenMP `critical` section wait times.

15.8. Objects

15.8.1. Enumerations

Enumerations are used to communicate options to many functions in Galacticus. All available enumerations, along with their members, are described below.

accretionMode

Description: Enumeration of accretion modes for the accrtionHalo class.
Provided by: module Accretion_Halos
Members: accretionModeTotal
accretionModeHot
accretionModeCold

adafEnergy

Description: Type of energy model to use for ADAFs.
Members: adafEnergyPureADAF
adafEnergyISCO

adafFieldEnhancement

Description: Type of field enhancement model to use for ADAFs.
Members: adafFieldEnhancementExponential
adafFieldEnhancementLinear

adafRadiativeEfficiencyType

Description: Type of radiative efficiency model to use for ADAFs.
Members: adafRadiativeEfficiencyTypeFixed
adafRadiativeEfficiencyTypeThinDisk

adafTable

Description: Enumeration of ADAF look-up tables.
Members: adafTablePowerJet
adafTableRateSpinUp

adafViscosity

Description: Type of viscosity model to use for ADAFs.
Members: adafViscosityFit
adafViscosityFixed

adjustElements

Description: Used to specify how elements should be adjusted when the metallicity of an abundances object is changed.
Provided by: module Abundances_Structure
Members: adjustElementsNone
adjustElementsReset
adjustElementsUpdate

angularMomentumSource

Description: Enumeration specifying the origin of angular momentum of cooling gas in the constant rotation class.
Members: angularMomentumSourceDarkMatter
angularMomentumSourceHotGas

bryanNorman1998Fit

Description: Specifies fit type for Bryan and Norman [1998] virial density contrast.
Members: `bryanNorman1998FitFlatUniverse`
`bryanNorman1998FitZeroLambda`

cambSpecies

Description: Particle species in CAMB.
Provided by: `module Interfaces_CAMB`
Members: `cambSpeciesDarkMatter`
`cambSpeciesBaryons`

comparison

Description: Comparison operators.
Provided by: `module Interface_Local_Group_DB`
Members: `comparisonEquals`
`comparisonGreaterThan`
`comparisonLessThan`

component

Description: Specifies components for linear growth factor.
Provided by: `module Linear_Growth`
Members: `componentDarkMatter`
`componentBaryons`
`componentRadiation`

componentType

Description: Used to specify the component(s) to be queried in galactic structure functions.
Provided by: `module Galactic_Structure_Options`
Members: `componentTypeAll`
`componentTypeDisk`
`componentTypeSpheroid`
`componentTypeHotHalo`
`componentTypeColdHalo`
`componentTypeDarkHalo`
`componentTypeBlackHole`

coordinateSystem

Description: Used to specify the coordinate system of the input coordinates in galactic structure functions.
Provided by: `module Galactic_Structure_Options`
Members: `coordinateSystemSpherical`
`coordinateSystemCylindrical`
`coordinateSystemCartesian`

cutOffWhen

Description: Specifies whether cooling is cut off before or after the given epoch.
Members: `cutOffWhenBefore`
`cutOffWhenAfter`

darkEnergySphericalCollapseEnergyFixedAt

Description: Enumeration of radii at which the energy of a spherical top-hat perturbation in a dark energy cosmology can be considered to be fixed.
Provided by: `module Spherical_Collapse_Matter_Dark_Energy`
Members: `darkEnergySphericalCollapseEnergyFixedAtTurnaround`
`darkEnergySphericalCollapseEnergyFixedAtVirialization`

dataType

Description: Used to specify the type of data being stored in a `mergerTreeData` structure metadata entry.
Provided by: `module Merger_Tree_Data_Structure`
Members: `dataTypeNull`
`dataTypeInteger`
`dataTypeDouble`
`dataTypeText`

deadlockStatus

Description: Specifies deadlock status during merger tree evolution.
Provided by: `module Merger_Trees_Evolve_Deadlock_Status`
Members: `deadlockStatusIsNotDeadlocked`
`deadlockStatusIsReporting`
`deadlockStatusIsSuspendable`
`deadlockStatusIsDeadlocked`

destinationMerger

Description: Enumeration of possible destinations for mass in mergers.
Provided by: `module Satellite_Merging_Mass_Movements`
Members: `destinationMergerUnmoved`
`destinationMergerDisk`
`destinationMergerSpheroid`

direction

Description: Used to specify integration directions output specifiers.
Provided by: `module Galactic_Structure_Radii_Definitions`
Members: `directionRadial`
`directionLineOfSight`
`directionLineOfSightInteriorAverage`
`directionLambdaR`

duttonMaccio2014DensityContrastMethod

Description: Enumeration of density contrast methods available in the `duttonMaccio2014` dark matter halo profile concentration class.
Members: `duttonMaccio2014DensityContrastMethodVirial`
`duttonMaccio2014DensityContrastMethodMean200`

duttonMaccio2014DensityProfileMethod

Description: Enumeration of density profile methods available in the `duttonMaccio2014` dark matter halo profile concentration class.
Members: `duttonMaccio2014DensityProfileMethodNFW`
`duttonMaccio2014DensityProfileMethodEinasto`

elementType

Description: Enumeration of extracted property element types.
Members: `elementTypeInteger`
`elementTypeDouble`

excursionSetRemap

Description: Specifies whether a remapping of the barrier should apply to cases where the excursion set is used for rate calculation, non-rate calculations, or both.
Provided by: `module Excursion_Sets_Barriers`
Members: `excursionSetRemapRates`
`excursionSetRemapNonRates`
`excursionSetRemapBoth`

extrapolationType

Description: Used to specify the type of extrapolation to use when interpolating in tables.
Provided by: `module Table_Labels`
Members: `extrapolationTypeExtrapolate`
`extrapolationTypeFix`
`extrapolationTypeAbort`
`extrapolationTypeZero`
`extrapolationTypePowerLaw`

fixedDensityType

Description: Specifies reference density type for fixed virial density contrast.
Members: `fixedDensityTypeCritical`
`fixedDensityTypeMean`

galacticComponent

Description: Specifies the galactic component for emission line calculations.
Members: `galacticComponentDisk`
`galacticComponentSpheroid`

galacticusParticleEpochType

Description: Particle epoch type enumerations.
Members: galacticusParticleEpochTypeTime
galacticusParticleEpochTypeExpansionFactor
galacticusParticleEpochTypeRedshift

gordon2003Sample

Description: Enumerates the samples available in the gordon2003 dust attenutation class.
Members: gordon2003SampleSMCbar
gordon2003SampleLMC

haloMassFunctionErrorModel

Description: Used to specify the error model to use for halo mass function likelihoods.
Members: haloMassFunctionErrorModelNone
haloMassFunctionErrorModelPowerLaw
haloMassFunctionErrorModelSphericalOverdensity
haloMassFunctionErrorModelTrenti2010

haloModelGalaxyType

Description: Used to specify galaxy types in the halo model.
Provided by: module Conditional_Mass_Functions
Members: haloModelGalaxyTypeAll
haloModelGalaxyTypeCentral
haloModelGalaxyTypeSatellite

haloModelTerm

Description: Enumeration of terms in the halo model.
Provided by: module Halo_Model_Power_Spectrum_Modifiers
Members: haloModelTermOneHalo
haloModelTermTwoHalo

hubbleUnits

Description: Specifies the units for the Hubble constant.
Provided by: module Cosmology_Parameters
Members: hubbleUnitsStandard
hubbleUnitsTime
hubbleUnitsLittleH

inputParameterErrorStatus

Description: Error status codes used by the input parameters module.
Provided by: module Input_Parameters
Members: inputParameterErrorStatusSuccess
inputParameterErrorStatusNotPresent
inputParameterErrorStatusParse
inputParameterErrorStatusEmptyValue
inputParameterErrorStatusAmbiguousValue

interpolant

Description: Specifies the different interpolants for emission line calculations.
Members: `interpolantMetallicity`
`interpolantDensity`
`interpolantHydrogen`
`interpolantHelium`
`interpolantOxygen`

ionizingContinuum

Description: Specifies the ionizing continuum for emission line calculations.
Members: `ionizingContinuumHydrogen`
`ionizingContinuumHelium`
`ionizingContinuumOxygen`

klypin2015DensityContrast

Description: Enumeration of density contrasts in the `klypin2015` dark matter halo profile concentration class.
Members: `klypin2015DensityContrastFixed`
`klypin2015DensityContrastVirial`

klypin2015FittingFunction

Description: Enumeration of fitting functions in the `klypin2015` dark matter halo profile concentration class.
Members: `klypin2015FittingFunctionEqn24`
`klypin2015FittingFunctionEqn25`

klypin2015Sample

Description: Enumeration of sample choices available in the `klypin2015` dark matter halo profile concentration class.
Members: `klypin2015SamplePlanck200CritRelaxedMass`
`klypin2015SamplePlanck200CritAllMass`
`klypin2015SamplePlanck200CritRelaxedVmax`
`klypin2015SamplePlanck200CritAllVmax`
`klypin2015SamplePlanckVirialRelaxedMass`
`klypin2015SamplePlanckVirialAllMass`
`klypin2015SamplePlanckVirialRelaxedVmax`
`klypin2015SamplePlanckVirialAllVmax`
`klypin2015SampleWmap7200CritRelaxedMass`
`klypin2015SampleWmap7200CritAllMass`
`klypin2015SampleWmap7200CritRelaxedVmax`
`klypin2015SampleWmap7VirialRelaxedMass`
`klypin2015SampleWmap7VirialAllMass`
`klypin2015SampleWmap7VirialRelaxedVmax`
`klypin2015SamplePlanck200CritAllMassUni`
`klypin2015SamplePlanck200CritRelaxedMassUni`
`klypin2015SamplePlanckVirialAllMassUni`
`klypin2015SamplePlanckVirialRelaxedMassUni`

latentIntegratorType

Description: Used to specify the type of latent variable integrator to use.
Members: latentIntegratorTypeGaussKronrod
latentIntegratorTypeTrapezoidal

massDistributionSymmetry

Description: Specifies the symmetry of massDistribution objects.
Provided by: module Mass_Distributions
Members: massDistributionSymmetryNone
massDistributionSymmetryCylindrical
massDistributionSymmetrySpherical

massType

Description: Used to specify the mass type(s) to be queried in galactic structure functions.
Provided by: module Galactic_Structure_Options
Members: massTypeAll
massTypeDark
massTypeBaryonic
massTypeGalactic
massTypeGaseous
massTypeStellar
massTypeBlackHole

mergerTreeBranchingBound

Description: Upper/lower bound labels used in merger tree branching calculations.
Provided by: module Merger_Tree_Branching
Members: mergerTreeBranchingBoundLower
mergerTreeBranchingBoundUpper

mergerTreeFormat

Description: Used to specify which output format to use for merger tree data.
Provided by: module Merger_Tree_Data_Structure
Members: mergerTreeFormatGalacticus
mergerTreeFormatIrate

metaDataType

Description: Used to specify the type of metadata being stored in a mergerTreeData structure.
Provided by: module Merger_Tree_Data_Structure
Members: metaDataTypeGeneric
metaDataTypeCosmology
metaDataTypeSimulation
metaDataTypeGroupFinder
metaDataTypeTreeBuilder
metaDataTypeProvenance

metallicityType

Description: Used to specify the metallicity scale when working with `abundances` objects.
Provided by: `module Abundances_Structure`
Members: `metallicityTypeLinearByMass`
`metallicityTypeLinearByNumber`
`metallicityTypeLogarithmicByMassSolar`
`metallicityTypeLogarithmicByNumberSolar`
`metallicityTypeLinearByMassSolar`
`metallicityTypeLinearByNumberSolar`
`metallicityTypeLogarithmicByNumberSolarPlus12`

nonAnalyticSolvers

Description: Used to specify the type of solution to use when no analytic solution is available.
Provided by: `module Dark_Matter_Profiles_Generic`
Members: `nonAnalyticSolversFallThrough`
`nonAnalyticSolversNumerical`

normalize

Description: Specifies normalization options for linear growth factor.
Provided by: `module Linear_Growth`
Members: `normalizeMatterDominated`
`normalizePresentDay`

openMPThreadBinding

Description: Used to specify how hooked functions are bound to OpenMP threads.
Provided by: `module Events_Hooks`
Members: `openMPThreadBindingNone`
`openMPThreadBindingAtLevel`
`openMPThreadBindingAllLevels`

outputAnalysisCovarianceModel

Description: Output analyses covariance models.
Provided by: `module Output_Analyses_Options`
Members: `outputAnalysisCovarianceModelPoisson`
`outputAnalysisCovarianceModelBinomial`

outputAnalysisPropertyQuantity

Description: Property quantities.
Provided by: `module Output_Analyses_Options`
Members: `outputAnalysisPropertyQuantityUnknown`
`outputAnalysisPropertyQuantityMass`
`outputAnalysisPropertyQuantityLuminosity`

outputAnalysisPropertyType

Description:	Property types.
Provided by:	module Output_Analyses_Options
Members:	outputAnalysisPropertyTypeLinear outputAnalysisPropertyTypeLog10 outputAnalysisPropertyTypeMagnitude outputAnalysisPropertyTypeUnknown

particulateKernel

Description:	Options for softening kernel in particulate.
Members:	particulateKernelDelta particulateKernelPlummer particulateKernelGadget

pathType

Description:	Enumeration of various paths used by Galacticus.
Provided by:	module Galacticus_Paths
Members:	pathTypeExec pathTypeDataStatic pathTypeDataDynamic

propertyType

Description: Used to specify properties in a `mergerTreeData` structure.
Provided by: `module Merger_Tree_Data_Structure`
Members:
`propertyTypeNull`
`propertyTypeTreeIndex`
`propertyTypeNodeIndex`
`propertyTypeDescendentIndex`
`propertyTypeHostIndex`
`propertyTypeRedshift`
`propertyTypeScaleFactor`
`propertyTypeNodeMass`
`propertyTypeNodeMass200Mean`
`propertyTypeNodeMass200Crit`
`propertyTypeParticleCount`
`propertyTypePositionX`
`propertyTypePositionY`
`propertyTypePositionZ`
`propertyTypeVelocityX`
`propertyTypeVelocityY`
`propertyTypeVelocityZ`
`propertyTypeSpinX`
`propertyTypeSpinY`
`propertyTypeSpinZ`
`propertyTypeSpin`
`propertyTypeAngularMomentumX`
`propertyTypeAngularMomentumY`
`propertyTypeAngularMomentumZ`
`propertyTypeAngularMomentum`
`propertyTypeSpecificAngularMomentumX`
`propertyTypeSpecificAngularMomentumY`
`propertyTypeSpecificAngularMomentumZ`
`propertyTypeSpecificAngularMomentum`
`propertyTypeHalfMassRadius`
`propertyTypeScaleRadius`
`propertyTypeParticleIndex`
`propertyTypeMostBoundParticleIndex`
`propertyTypeSnapshot`
`propertyTypeTreeWeight`
`propertyTypeVelocityMaximum`
`propertyTypeVelocityDispersion`

pushType

Description: Cross-tree event type enumeration.
Members:
`pushTypeBranchJump`
`pushTypeSubhaloPromotion`

radiusFixed

Description: Enumerates the possible definitions of radius used by the "fixed" galactic structure solver.
Members: radiusFixedVirial
radiusFixedTurnaround

radiusType

Description: Used to specify radii types used in output specifiers.
Provided by: module Galactic_Structure_Radii_Definitions
Members: radiusTypeRadius
radiusTypeVirialRadius
radiusTypeDarkMatterScaleRadius
radiusTypeDiskRadius
radiusTypeSpheroidRadius
radiusTypeDiskHalfMassRadius
radiusTypeSpheroidHalfMassRadius
radiusTypeGalacticMassFraction
radiusTypeGalacticLightFraction

randomSampleCountType

Description: Used to specify the type of random sample count.
Members: randomSampleCountTypeFixed
randomSampleCountTypeMultiplicative

rangeExpand

Description: Used to specify the way in which the bracketing range should be expanded when searching for roots using a rootFinder object.
Provided by: module Root_Finder
Members: rangeExpandNull
rangeExpandAdditive
rangeExpandMultiplicative

rangeExpandSignExpect

Description: Used to specify the expected sign of the root function when searching for roots using a rootFinder object.
Provided by: module Root_Finder
Members: rangeExpandSignExpectNegative
rangeExpandSignExpectNone
rangeExpandSignExpectPositive

readNodeReachability

Description: Node reachability status.
Members: readNodeReachabilityUnreachable
readNodeReachabilityReachable

readSubhaloAngularMomentaMethod

Description: Subhalo angular momentum methods.
Members: `readSubhaloAngularMomentaMethodScale`
`readSubhaloAngularMomentaMethodSummation`

recombinationCase

Description: Enumeration of radiative recombination cases (A or B).
Provided by: `module Atomic_Rates_Recombination_Radiative`
Members: `recombinationCaseA`
`recombinationCaseB`

replicantAction

Description: Used to specify type of action required from the replicant method.
Members: `replicantActionCount`
`replicantActionAny`
`replicantActionInstance`

satelliteBoundMassInitializeType

Description: Specify how to initialize the bound mass of a satellite halo.
Provided by: `module Node_Component_Satellite_Orbiting`
Members: `satelliteBoundMassInitializeTypeBasicMass`
`satelliteBoundMassInitializeTypeMaximumRadius`
`satelliteBoundMassInitializeTypeDensityContrast`

satelliteStatusDiscriminator

Description: Enumeration of possible discriminators for satellite orphan status.
Members: `satelliteStatusDiscriminatorBoundMass`
`satelliteStatusDiscriminatorPosition`

sedFitDustType

Description: Used to specify the type of dust model to use in SED fitting likelihoods.
Members: `sedFitDustTypeNull`
`sedFitDustTypeCharlotFall2000`
`sedFitDustTypeCardelli1989`
`sedFitDustTypeGordon2003`
`sedFitDustTypeCalzetti2000`
`sedFitDustTypeWittGordon2000`

sedFitStartTime

Description: Used to specify the type of start time to use in SED fitting likelihoods.
Members: `sedFitStartTimeTime`
`sedFitStartTimeAge`

selection

Description: Options for selection of nodes to particulate.
Members: selectionAll
selectionHosts
selectionSatellites

setOperator

Description: Set operators.
Provided by: module Interface_Local_Group_DB
Members: setOperatorIntersection
setOperatorUnion
setOperatorRelativeComplement

snapshotSpacing

Description: Specifies the spacing of snapshots when regridding merger trees.
Members: snapshotSpacingLinear
snapshotSpacingLogarithmic
snapshotSpacingLogCriticalOverdensity
snapshotSpacingMillennium
snapshotSpacingList

spinDistributionType

Description: Used to specify the type of intrinsic spin distribution.
Members: spinDistributionTypeLogNormal
spinDistributionTypeBett2007

standardODEAlgorithm

Description: Used to specify the type of ODE algorithm to use.
Members: standardODEAlgorithmRungeKuttaCashKarp
standardODEAlgorithmRungeKuttaSecondOrder
standardODEAlgorithmRungeKutta
standardODEAlgorithmRungeKuttaFehlberg
standardODEAlgorithmRungeKuttaPrinceDormand
standardODEAlgorithmMultistepAdams
standardODEAlgorithmBulirschStoer
standardODEAlgorithmBdf

sussingBadValueTest

Description: Bad value test options.
Members: sussingBadValueTestLessThan
sussingBadValueTestGreaterThan

sussingHaloFormat

Description: Halo file formats.
Members: sussingHaloFormatOld
sussingHaloFormatNew
sussingHaloFormatAll

sussingMassOption

Description: Halo mass definitions.
Members: sussingMassOptionDefault
sussingMassOptionFoF
sussingMassOption200Mean
sussingMassOption200Crit
sussingMassOptionTopHat

tableType

Description: Enumeration of table types.
Provided by: module Tables
Members: tableTypeLinearLinear1D
tableTypeLogarithmicLinear1D

test

Description: Statuses for unit tests.
Provided by: module Unit_Tests
Members: testPassed
testFailed
testSkipped

tinker2008Parameter

Description: Enumeration of parameters for the tinker2008 halo mass function class.
Members: tinker2008ParameterA
tinker2008ParameterB
tinker2008ParameterC
tinker2008ParameterNormalization

treeBuild

Description: Enumeration of tree building status.
Members: treeBuildSuccess
treeBuildFailureTolerance
treeBuildFailureStructure
treeBuildFailureGeneric

treeStatistic

Description: Enumeration of tasks to be performed during a tree walk.
Members: treeStatisticNodeCount
treeStatisticEndNodeCount

`units`

Description: Used to specify the type of units being stored in a `mergerTreeData` structure.
Provided by: `module Merger_Tree_Data_Structure`
Members: `unitsMass`
`unitsLength`
`unitsTime`
`unitsVelocity`

`weightBy`

Description: Used to specify by which quantity to weight the results in galactic structure functions.
Provided by: `module Galactic_Structure_Options`
Members: `weightByMass`
`weightByLuminosity`

`wittGordon2000Model`

Description: Enumerates the models available in the `wittGorden2000` dust attenutation class.
Members: `wittGordon2000ModelMilkyWayShellTau3`
`wittGordon2000ModelSMCShellTau3`

`structureErrorCode`

Description: Error codes for galactic structure functions.
Provided by: `module Galactic_Structure_Options`
Members: `structureErrorCodeSuccess`
`structureErrorCodeInfinite`

15.8.2. Object Methods

The type of each method, and the type and names of its arguments are specified for each method of each object. Types are shown in red, enclosed by angle brackets, with a "`*`" indicating a pointer. A `<void>` type indicates a subroutine. Blue arrows after each argument show the argument intent: ← implies `intent(in)`, → implies `intent(out)`, and ↔ implies `intent(inout)`.

`abundances`

`<type(abundances)>` `add(<type(abundances)>` `abundances2`→`)` Add two abundances.

`<void>` `builder(<*type(node)>` `abundancesDefinition`→`)` Build an abundances object from a provided XML description.

`<void>` `deserialize(<double(:)>` `historyArray`→`)` Deserialize an abundances object from an array.

`<void>` `destroy()` Destroy an abundances object.

`<type(abundances)>` `divide(<double>` `divisor`→`)` Divide an abundance by a scalar.

`<void>` `dump()` Dump an abundances object.

`<void>` `dumpRaw(<integer>` `fileHandle`→`)` Dump an abundances object to binary.

`<double>` `heliumMassFraction()` Returns the helium fraction by mass.

<double> heliumNumberFraction() Returns the helium fraction by number.

<double> hydrogenMassFraction() Returns the hydrogen fraction by mass.

<double> hydrogenNumberFraction() Returns the hydrogen fraction by number.

<void> increment(<type(abundances)> addAbundances→) Increment an abundances object.

<logical> isZero() Return true if an abundances object is zero.

<void> massToMassFraction(<double> mass→) Converts abundance masses to mass fractions by dividing by the given mass while ensuring that fractions are in the range 0–1.

<double> metallicity(<metallicityType> [metallicityType]→) Returns the metallicity.

<void> metallicitySet(<double> metallicity→, <metallicityType> [metallicityType]→, <adjustElements> [adjustElements]→, <integer> [abundanceIndex]→) Sets the metallicity to metallicity.

<type(abundances)> multiply(<double> multiplier→) Multiply an abundance by a scalar.

<integer(c_size_t) > nonStaticSizeOf() Returns the size of any non-static components of the type.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(:,:)> integerBuffer↔, <integer>doubleProperty↔, <integer> doubleBufferCount↔, <double(:,:)> doubleBuffer↔, <double> time→, <integer> instance→) Store an abundances object in the output buffers.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double> time→, <integer> instance→) Specify the count of an abundances object for output.

<void> outputNames(<integer> integerProperty↔, <char[*](:)> integerPropertyNames↔, <char[*](:)> integerPropertyComments↔, <double(:)> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <char[*](:)> doublePropertyNames↔, <char[*](:)> doublePropertyComments↔, <double(:)> doublePropertyUnitsSI↔, <double> time→, <integer> instance→) Specify the names of abundance object properties for output.

<void> postOutput(<double> time→) Store an abundances object in the output buffers.

<void> readRaw(<integer> fileHandle→) Read an abundances object from binary.

<void> reset() Reset an abundances object.

<void> serialize(<double(:)> historyArray←) Serialize an abundances object to an array.

<integer> serializeCount() Return a count of the number of properties in a serialized abundances object.

<void> setToUnity() Set an abundances object to unity.

<type(abundances)> subtract(<type(abundances)> abundances2→) Subtract one abundance from another.

accretionDisksADAF

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

<double> angularMomentum(<double> spinBlackHole→, <double> radius→) Return the dimensionless angular momentum of the ADAF at a given radius.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((accretionDisksClass))\textgreater} destination\arginout)
Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision efficiencyRadiative(\textcolor{red}{\textless class((nodeComponentBlackHole))\textgrea
blackHole\arginout,\textcolor{red}{\textless double precision\textgreater} accretionRateMass\argin)
Returns the radiative efficiency of the accretion disk.

<double> enthalpy(<double> spinBlackHole→, <double> radius→) Return the dimensionless enthalpy of the ADAF at a given radius.

<double> enthalpyAngularMomentumProduct(<double> spinBlackHole→, <double> radius→) Return the product of dimensionless enthalpy and angular momentum of the ADAF at a given radius.

<double> fieldEnhancement(<double> spinBlackHole→, <double> radius→) Return the magnetic field enhancement factor in the ADAF at the given radius.

<double> fluidAngularVelocity(<double> spinBlackHole→, <double> radius→) Return the dimensionless angular velocity of the ADAF fluid at the given radius.

<double> gamma(<double> spinBlackHole→, <double> radius→) Return the relativistic boost factor in the ADAF at a given radius.

<double> gammaAzimuthal(<double> spinBlackHole→, <double> radius→) Return the azimuthal part of the relativistic boost factor in the ADAF at a given radius.

<double> gammaRadial(<double> spinBlackHole→, <double> radius→) Return the radial part of the relativistic boost factor in the ADAF at a given radius.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<double> height(<double> spinBlackHole→, <double> radius→) Return the dimensionless height of the ADAF at a given radius.

<logical> isDefault() Return true if this is the default object of this class.

<double> jetPowerBlackHole(<double> spinBlackHole→, <double> radius→) Return the power of the jet launched by the black hole for the ADAF.

<double> jetPowerDisk(<double> spinBlackHole→, <double> radius→) Return the power of the jet launched from the disk for the ADAF.

<double> jetPowerDiskFromBlackHole(<double> spinBlackHole→, <double> radius→) Return the power of the jet launched from the disk which is derived frm the black hole.

type(varying_string) objectType() Return the type of the object.

double precision powerJet(\textcolor{red}{\textless class((nodeComponentBlackHole))\textgreater} blackHole\arginout,\textcolor{red}{\textless double precision\textgreater} accretionRateMass\argin) Returns the power of the jet launched by the accretion disk in units of $M_\odot$ (km/s)^2 Gyr^{-1}.

double precision rateSpinUp(\textcolor{red}{\textless class((nodeComponentBlackHole))\textgreater} blackHole\arginout,\textcolor{red}{\textless double precision\textgreater} accretionRateMass\argin) Returns the spin-up rate of the black hole due to accretion from the accretion disk.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

<double> temperature(<double> spinBlackHole→, <double> radius→) Return the dimensionless temperature of the ADAF at a given radius.

<double> velocity(<double> spinBlackHole→, <double> radius→) Return the dimensionless velocity of the ADAF at a given radius.

<double> viscosityParameter(<double> spinBlackHole→) Return the viscosity parameter, α, in the ADAF.

accretionDisksClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((accretionDisksClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision efficiencyRadiative(\textcolor{red}{\textless class((nodeComponentBlackHole))\textgrea blackHole\arginout,\textcolor{red}{\textless double precision\textgreater} accretionRateMass\argin) Returns the radiative efficiency of the accretion disk.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision powerJet(\textcolor{red}{\textless class((nodeComponentBlackHole))\textgreater} blackHole\arginout,\textcolor{red}{\textless double precision\textgreater} accretionRateMass\argin)
Returns the power of the jet launched by the accretion disk in units of $M_\odot$ (km/s)^2 Gyr^{-1}.

double precision rateSpinUp(\textcolor{red}{\textless class((nodeComponentBlackHole))\textgreater} blackHole\arginout,\textcolor{red}{\textless double precision\textgreater} accretionRateMass\argin)
Returns the spin-up rate of the black hole due to accretion from the accretion disk.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

accretionDisksEddingtonLimited

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((accretionDisksClass))\textgreater} destination\arginout)
Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision efficiencyRadiative(\textcolor{red}{\textless class((nodeComponentBlackHole))\textgrea blackHole\arginout,\textcolor{red}{\textless double precision\textgreater} accretionRateMass\argin)
Returns the radiative efficiency of the accretion disk.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision powerJet(\textcolor{red}{\textless class((nodeComponentBlackHole))\textgreater} blackHole\arginout,\textcolor{red}{\textless double precision\textgreater} accretionRateMass\argin) Returns the power of the jet launched by the accretion disk in units of $M_\odot$ (km/s)^2 Gyr^{-1}.

double precision rateSpinUp(\textcolor{red}{\textless class((nodeComponentBlackHole))\textgreater} blackHole\arginout,\textcolor{red}{\textless double precision\textgreater} accretionRateMass\argin) Returns the spin-up rate of the black hole due to accretion from the accretion disk.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

accretionDiskSpectraClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((accretionDiskSpectraClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\textcolor{red}{\textless logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDigest\argin) Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

double precision spectrum(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} wavelength\argin) Returns the spectrum (in units of $L_\odot$ Hz^{-1}) of the accretion disk at the given wavelength (in units of Å) for node.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

accretionDiskSpectraFile

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((accretionDiskSpectraClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

void loadFile(<character(len=*)> fileName→) Load a file of AGN spectra.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

double precision spectrum(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textco double precision\textgreater} wavelength\argin) Returns the spectrum (in units of $L_\odot$ Hz^{-1}) of the accretion disk at the given wavelength (in units of Å) for node.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

accretionDiskSpectraHopkins2007

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`<void> buildFile()` Build the tabulation file containing AGN spectra.

`void deepCopy(\textcolor{red}{\textless class((accretionDiskSpectraClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`void loadFile(<character(len=*)> fileName→)` Load a file of AGN spectra.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`double precision spectrum(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textco double precision\textgreater} wavelength\argin)` Returns the spectrum (in units of $L_\odot$ Hz^{-1}) of the accretion disk at the given wavelength (in units of Å) for `node`.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

accretionDisksShakuraSunyaev

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((accretionDisksClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision efficiencyRadiative(\textcolor{red}{\textless class((nodeComponentBlackHole))\textgrea blackHole\arginout,\textcolor{red}{\textless double precision\textgreater} accretionRateMass\argin) Returns the radiative efficiency of the accretion disk.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision powerJet(\textcolor{red}{\textless class((nodeComponentBlackHole))\textgreater} blackHole\arginout,\textcolor{red}{\textless double precision\textgreater} accretionRateMass\argin) Returns the power of the jet launched by the accretion disk in units of $M_\odot$ (km/s)^2 Gyr^{-1}.

double precision rateSpinUp(\textcolor{red}{\textless class((nodeComponentBlackHole))\textgreater} blackHole\arginout,\textcolor{red}{\textless double precision\textgreater} accretionRateMass\argin) Returns the spin-up rate of the black hole due to accretion from the accretion disk.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

accretionDisksSwitched

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((accretionDisksClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision efficiencyRadiative(\textcolor{red}{\textless class((nodeComponentBlackHole))\textgrea blackHole\arginout,\textcolor{red}{\textless double precision\textgreater} accretionRateMass\argin) Returns the radiative efficiency of the accretion disk.

<double> efficiencyRadiativeScalingADAF(<class(nodeComponentBlackHole)> blackHole→, <double> accretionRateMass→) Return the scaling of radiative efficiency of the ADAF component in a switched accretion disk.

<double> fractionADAF(<class(nodeComponentBlackHole)> blackHole→, <double> accretionRateMass→) Return the fraction of the accretion flow to be represented as an ADAF.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision powerJet(\textcolor{red}{\textless class((nodeComponentBlackHole))\textgreater} blackHole\arginout,\textcolor{red}{\textless double precision\textgreater} accretionRateMass\argin) Returns the power of the jet launched by the accretion disk in units of $M_\odot$ (km/s)^2 Gyr^{-1}.

double precision rateSpinUp(\textcolor{red}{\textless class((nodeComponentBlackHole))\textgreater} blackHole\arginout,\textcolor{red}{\textless double precision\textgreater} accretionRateMass\argin) Returns the spin-up rate of the black hole due to accretion from the accretion disk.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

accretionHaloBertschinger

double precision accretedMass(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\te integer\textgreater} accretionMode\argin) Returns the mass (in units of $M_\odot$) of accreted from the IGM onto node in the given accretionMode. Used to initialize nodes.

type(chemicalAbundances) accretedMassChemicals(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater} accretionMode\argin) Returns the mass of chemicals (in units of $M_\odot$) of accreted from the IGM onto node in the given accretionMode. Used to initialize nodes.

type(abundances) accretedMassMetals(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater} accretionMode\argin) Returns the mass of metals (in units of $M_\odot$) of accreted from the IGM onto node in the given accretionMode. Used to initialize nodes.

`double precision accretionRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater} accretionMode\argin)` Returns the rate (in units of $M_\odot$ Gyr^{-1}) of accretion of mass from the IGM onto `node` in the given `accretionMode`.

`type(chemicalAbundances) accretionRateChemicals(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater} accretionMode\argin)` Returns the rate (in units of $M_\odot$ Gyr^{-1}) of accretion of chemicals from the IGM onto `node` in the given `accretionMode`.

`type(abundances) accretionRateMetals(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater} accretionMode\argin)` Returns the rate (in units of $M_\odot$ Gyr^{-1}) of accretion of metals from the IGM onto `node` in the given `accretionMode`.

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`logical branchHasBaryons(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Returns `true` if this tree branch may accrete baryons, and `false` otherwise.

`void deepCopy(\textcolor{red}{\textless class((accretionHaloClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision failedAccretedMass(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater} accretionMode\argin)` Returns the mass (in units of $M_\odot$) of that failed to accrete from the IGM onto `node` in the given `accretionMode`. Used to initialize nodes.

`double precision failedAccretionRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater} accretionMode\argin)` Returns the rate (in units of $M_\odot$ Gyr^{-1}) of failed accretion of mass from the IGM onto `node` in the given `accretionMode`.

`double precision failedFraction(<type(treeNode)> *node↔)` Returns the fraction of potential accretion onto a halo from the IGM which fails.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision velocityScale(<type(treeNode)> *node↔) Returns the velocity scale to use for node.

accretionHaloClass

double precision accretedMass(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\te integer\textgreater} accretionMode\argin) Returns the mass (in units of $M_\odot$) of accreted from the IGM onto node in the given accretionMode. Used to initialize nodes.

type(chemicalAbundances) accretedMassChemicals(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater} accretionMode\argin) Returns the mass of chemicals (in units of $M_\odot$) of accreted from the IGM onto node in the given accretionMode. Used to initialize nodes.

type(abundances) accretedMassMetals(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater} accretionMode\argin) Returns the mass of metals (in units of $M_\odot$) of accreted from the IGM onto node in the given accretionMode. Used to initialize nodes.

double precision accretionRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater} accretionMode\argin) Returns the rate (in units of $M_\odot$ Gyr^{-1}) of accretion of mass from the IGM onto node in the given accretionMode.

type(chemicalAbundances) accretionRateChemicals(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater} accretionMode\argin) Returns the rate (in units of $M_\odot$ Gyr^{-1}) of accretion of chemicals from the IGM onto node in the given accretionMode.

type(abundances) accretionRateMetals(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater} accretionMode\argin) Returns the rate (in units of $M_\odot$ Gyr^{-1}) of accretion of metals from the IGM onto node in the given accretionMode.

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

logical branchHasBaryons(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns true if this tree branch may accrete baryons, and false otherwise.

void deepCopy(\textcolor{red}{\textless class((accretionHaloClass))\textgreater} destination\arginout) Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision failedAccretedMass(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater} accretionMode\argin)` Returns the mass (in units of $M_\odot$) of that failed to accrete from the IGM onto `node` in the given `accretionMode`. Used to initialize nodes.

`double precision failedAccretionRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater} accretionMode\argin)` Returns the rate (in units of $M_\odot$ Gyr^{-1}) of failed accretion of mass from the IGM onto `node` in the given `accretionMode`.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

accretionHaloColdMode

`double precision accretedMass(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\te integer\textgreater} accretionMode\argin)` Returns the mass (in units of $M_\odot$) of accreted from the IGM onto `node` in the given `accretionMode`. Used to initialize nodes.

`type(chemicalAbundances) accretedMassChemicals(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater} accretionMode\argin)` Returns the mass of chemicals (in units of $M_\odot$) of accreted from the IGM onto `node` in the given `accretionMode`. Used to initialize nodes.

`type(abundances) accretedMassMetals(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater} accretionMode\argin)` Returns the mass of metals (in units of $M_\odot$) of accreted from the IGM onto `node` in the given `accretionMode`. Used to initialize nodes.

`double precision accretionRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater} accretionMode\argin)` Returns the rate (in units of $M_\odot$ Gyr^{-1}) of accretion of mass from the IGM onto `node` in the given `accretionMode`.

type(chemicalAbundances) accretionRateChemicals(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater} accretionMode\argin) Returns the rate (in units of $M_\odot$ Gyr^{-1}) of accretion of chemicals from the IGM onto node in the given accretionMode.

type(abundances) accretionRateMetals(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater} accretionMode\argin) Returns the rate (in units of $M_\odot$ Gyr^{-1}) of accretion of metals from the IGM onto node in the given accretionMode.

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

logical branchHasBaryons(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns true if this tree branch may accrete baryons, and false otherwise.

<void> calculationReset(<type(table)> node↔) Reset memoized calculations.

<type(chemicalAbundances)> chemicalMasses(<type(treeNode)> *node↔, <double>massAccreted→, <integer>accretionMode→) Returns the total accretion rate from the IGM onto a halo (including dark matter).

<double> coldModeFraction(<type(treeNode)> *node↔, <integer>accretionMode→) Returns the total accretion rate from the IGM onto a halo (including dark matter).

void deepCopy(\textcolor{red}{\textless class((accretionHaloClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision failedAccretedMass(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater} accretionMode\argin) Returns the mass (in units of $M_\odot$) of that failed to accrete from the IGM onto node in the given accretionMode. Used to initialize nodes.

double precision failedAccretionRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater} accretionMode\argin) Returns the rate (in units of $M_\odot$ Gyr^{-1}) of failed accretion of mass from the IGM onto node in the given accretionMode.

double precision failedFraction(<type(treeNode)> *node↔) Returns the fraction of potential accretion onto a halo from the IGM which fails.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision velocityScale(<type(treeNode)> *node↔) Returns the velocity scale to use for node.

accretionHaloNaozBarkana2007

double precision accretedMass(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\te integer\textgreater} accretionMode\argin) Returns the mass (in units of $M_\odot$) of accreted from the IGM onto node in the given accretionMode. Used to initialize nodes.

type(chemicalAbundances) accretedMassChemicals(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater} accretionMode\argin) Returns the mass of chemicals (in units of $M_\odot$) of accreted from the IGM onto node in the given accretionMode. Used to initialize nodes.

type(abundances) accretedMassMetals(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater} accretionMode\argin) Returns the mass of metals (in units of $M_\odot$) of accreted from the IGM onto node in the given accretionMode. Used to initialize nodes.

double precision accretionRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater} accretionMode\argin) Returns the rate (in units of $M_\odot$ Gyr^{-1}) of accretion of mass from the IGM onto node in the given accretionMode.

type(chemicalAbundances) accretionRateChemicals(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater} accretionMode\argin) Returns the rate (in units of $M_\odot$ Gyr^{-1}) of accretion of chemicals from the IGM onto node in the given accretionMode.

type(abundances) accretionRateMetals(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater} accretionMode\argin) Returns the rate (in units of $M_\odot$ Gyr^{-1}) of accretion of metals from the IGM onto node in the given accretionMode.

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

`logical branchHasBaryons(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Returns `true` if this tree branch may accrete baryons, and `false` otherwise.

`<void> calculationReset(<type(table)> node↔)` Reset memoized calculations.

`void deepCopy(\textcolor{red}{\textless class((accretionHaloClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision failedAccretedMass(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater} accretionMode\argin)` Returns the mass (in units of $M_\odot$) of that failed to accrete from the IGM onto `node` in the given `accretionMode`. Used to initialize nodes.

`double precision failedAccretionRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater} accretionMode\argin)` Returns the rate (in units of $M_\odot$ Gyr^{-1}) of failed accretion of mass from the IGM onto `node` in the given `accretionMode`.

`double precision failedFraction(<type(treeNode)> *node↔)` Returns the fraction of potential accretion onto a halo from the IGM which fails.

`double precision filteredFraction(<type(treeNode)> *node↔)` Returns the fraction of potential accretion onto a halo from the IGM which succeeded.

`double precision filteredFractionCompute(<double> massHalo→, <double> massFiltering→)` Returns the fraction of potential accretion onto a halo from the IGM which succeeded given the halo and filtering masses.

`double precision filteredFractionRate(<type(treeNode)> *node↔)` Returns the fraction of potential accretion rate onto a halo from the IGM which succeeds.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision velocityScale(<type(treeNode)> *node↔)` Returns the velocity scale to use for `node`.

accretionHaloSimple

`double precision accretedMass(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\te integer\textgreater} accretionMode\argin)` Returns the mass (in units of $M_\odot$) of accreted from the IGM onto `node` in the given `accretionMode`. Used to initialize nodes.

`type(chemicalAbundances) accretedMassChemicals(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater} accretionMode\argin)` Returns the mass of chemicals (in units of $M_\odot$) of accreted from the IGM onto `node` in the given `accretionMode`. Used to initialize nodes.

`type(abundances) accretedMassMetals(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater} accretionMode\argin)` Returns the mass of metals (in units of $M_\odot$) of accreted from the IGM onto `node` in the given `accretionMode`. Used to initialize nodes.

`double precision accretionRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater} accretionMode\argin)` Returns the rate (in units of $M_\odot$ Gyr^{-1}) of accretion of mass from the IGM onto `node` in the given `accretionMode`.

`type(chemicalAbundances) accretionRateChemicals(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater} accretionMode\argin)` Returns the rate (in units of $M_\odot$ Gyr^{-1}) of accretion of chemicals from the IGM onto `node` in the given `accretionMode`.

`type(abundances) accretionRateMetals(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater} accretionMode\argin)` Returns the rate (in units of $M_\odot$ Gyr^{-1}) of accretion of metals from the IGM onto `node` in the given `accretionMode`.

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`logical branchHasBaryons(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Returns `true` if this tree branch may accrete baryons, and `false` otherwise.

`void deepCopy(\textcolor{red}{\textless class((accretionHaloClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision failedAccretedMass(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater} accretionMode\argin)` Returns the mass (in units of $M_\odot$) of that failed to accrete from the IGM onto `node` in the given `accretionMode`. Used to initialize nodes.

`double precision failedAccretionRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater} accretionMode\argin)` Returns the rate (in units of $M_\odot$ Gyr^{-1}) of failed accretion of mass from the IGM onto `node` in the given `accretionMode`.

`double precision failedFraction(<type(treeNode)> *node↔)` Returns the fraction of potential accretion onto a halo from the IGM which fails.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl` `type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-` `size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles` `type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-` `size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision velocityScale(<type(treeNode)> *node↔)` Returns the velocity scale to use for `node`.

accretionHaloTotalBertschinger

`double precision accretedMass(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Return the total accreted mass in the given `node`.

`double precision accretionRate(\textcolor{red}{\textless type((treeNode))\textgreater}` `node\arginout)` Return the total accretion rate onto the given `node`.

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\` `character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((accretionHaloTotalClass))\textgreater} destination\argin` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex` `logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

accretionHaloTotalClass

double precision accretedMass(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Return the total accreted mass in the given node.

double precision accretionRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Return the total accretion rate onto the given node.

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((accretionHaloTotalClass))\textgreater} destination\argin Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

accretionHaloTotalSimple

`double precision accretedMass(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Return the total accreted mass in the given `node`.

`double precision accretionRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Return the total accretion rate onto the given `node`.

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((accretionHaloTotalClass))\textgreater} destination\argin` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

accretionHaloZero

`double precision accretedMass(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\te integer\textgreater} accretionMode\argin)` Returns the mass (in units of $M_\odot$) of accreted from the IGM onto `node` in the given `accretionMode`. Used to initialize nodes.

`type(chemicalAbundances) accretedMassChemicals(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater} accretionMode\argin)` Returns the mass of chemicals (in units of $M_\odot$) of accreted from the IGM onto `node` in the given `accretionMode`. Used to initialize nodes.

`type(abundances) accretedMassMetals(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater} accretionMode\argin)` Returns the mass of metals (in units of $M_\odot$) of accreted from the IGM onto `node` in the given `accretionMode`. Used to initialize nodes.

`double precision accretionRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater} accretionMode\argin)` Returns the rate (in units of $M_\odot$ Gyr^{-1}) of accretion of mass from the IGM onto `node` in the given `accretionMode`.

`type(chemicalAbundances) accretionRateChemicals(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater} accretionMode\argin)` Returns the rate (in units of $M_\odot$ Gyr^{-1}) of accretion of chemicals from the IGM onto `node` in the given `accretionMode`.

`type(abundances) accretionRateMetals(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater} accretionMode\argin)` Returns the rate (in units of $M_\odot$ Gyr^{-1}) of accretion of metals from the IGM onto `node` in the given `accretionMode`.

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`logical branchHasBaryons(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Returns `true` if this tree branch may accrete baryons, and `false` otherwise.

`void deepCopy(\textcolor{red}{\textless class((accretionHaloClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision failedAccretedMass(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater} accretionMode\argin)` Returns the mass (in units of $M_\odot$) of that failed to accrete from the IGM onto `node` in the given `accretionMode`. Used to initialize nodes.

`double precision failedAccretionRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater} accretionMode\argin)` Returns the rate (in units of $M_\odot$ Gyr^{-1}) of failed accretion of mass from the IGM onto `node` in the given `accretionMode`.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

atomicCrossSectionIonizationPhotoClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

double precision crossSection(\textcolor{red}{\textless integer\textgreater} atomicNumber\argin,\textco integer\textgreater} ionizationState\argin,\textcolor{red}{\textless integer\textgreater} shellNumber\argin,\textcolor{red}{\textless double precision\textgreater} wavelength\argin) Returns the cross-section for photoionization (in units of cm^2) for a given atom in a given ionization state at the specified wavelength (given in units of Å).

void deepCopy(\textcolor{red}{\textless class((atomicCrossSectionIonizationPhotoClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

atomicCrossSectionIonizationPhotoVerner

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`double precision crossSection(\textcolor{red}{\textless integer\textgreater} atomicNumber\argin,\textco integer\textgreater} ionizationState\argin,\textcolor{red}{\textless integer\textgreater} shellNumber\argin,\textcolor{red}{\textless double precision\textgreater} wavelength\argin)` Returns the cross-section for photoionization (in units of cm^2) for a given atom in a given ionization state at the specified `wavelength` (given in units of Å).

`void deepCopy(\textcolor{red}{\textless class((atomicCrossSectionIonizationPhotoClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

atomicExcitationRateCollisionalClass

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

double precision coolingRate(\textcolor{red}{\textless integer\textgreater} atomicNumber\argin,\textcol integer\textgreater} electronNumber\argin,\textcolor{red}{\textless double precision\textgreater} temperature\argin) Return the collisional excitation cooling rate , in units of J/m^3/s, for ion of given atomicNumber and electronNumber at temperature T (in Kelvin).

void deepCopy(\textcolor{red}{\textless class((atomicExcitationRateCollisionalClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

atomicExcitationRateCollisionalScholzWalters1991

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

double precision coolingRate(\textcolor{red}{\textless integer\textgreater} atomicNumber\argin,\textcol integer\textgreater} electronNumber\argin,\textcolor{red}{\textless double precision\textgreater} temperature\argin) Return the collisional excitation cooling rate , in units of J/m^3/s, for ion of given atomicNumber and electronNumber at temperature T (in Kelvin).

void deepCopy(\textcolor{red}{\textless class((atomicExcitationRateCollisionalClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

atomicIonizationPotentialClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((atomicIonizationPotentialClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision potential(\textcolor{red}{\textless integer\textgreater} atomicNumber\argin,\textcolo integer\textgreater} electronNumber\argin) Returns the ionization potential (in units of eV) for a given atom in a given ionization state.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

atomicIonizationPotentialVerner

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((atomicIonizationPotentialClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision potential(\textcolor{red}{\textless integer\textgreater} atomicNumber\argin,\textcolor
integer\textgreater} electronNumber\argin) Returns the ionization potential (in units of eV) for a given atom in a given ionization state.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

atomicIonizationRateCollisionalClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((atomicIonizationRateCollisionalClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision rate(\textcolor{red}{\textless integer\textgreater} atomicNumber\argin,\textcolor{red}
integer\textgreater} ionizationState\argin,\textcolor{red}{\textless double precision\textgreater}
temperature\argin) Returns the radiative recombination rate.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

atomicIonizationRateCollisionalVerner1996

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((atomicIonizationRateCollisionalClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision rate(\textcolor{red}{\textless integer\textgreater} atomicNumber\argin,\textcolor{red}
integer\textgreater} ionizationState\argin,\textcolor{red}{\textless double precision\textgreater}
temperature\argin) Returns the radiative recombination rate.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

atomicRecombinationRateDielectronicArnaud1985

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((atomicRecombinationRateDielectronicClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision rate(\textcolor{red}{\textless integer\textgreater} atomicNumber\argin,\textcolor{red} integer\textgreater} electronNumber\argin,\textcolor{red}{\textless double precision\textgreater} temperature\argin) Return the dielectroninc recombination rate (in units of cm^3 s^{-1}) for the ion of given atomicNumber and electronNumber at the given temperature (in Kelvin).

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

atomicRecombinationRateDielectronicClass

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((atomicRecombinationRateDielectronicClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`double precision rate(\textcolor{red}{\textless integer\textgreater} atomicNumber\argin,\textcolor{red} integer\textgreater} electronNumber\argin,\textcolor{red}{\textless double precision\textgreater} temperature\argin)` Return the dielectroninc recombination rate (in units of $\mathrm{cm}^3\ \mathrm{s}^{-1}$) for the ion of given `atomicNumber` and `electronNumber` at the given `temperature` (in Kelvin).

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

atomicRecombinationRateRadiativeClass

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((atomicRecombinationRateRadiativeClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision rate(\textcolor{red}{\textless integer\textgreater} atomicNumber\argin,\textcolor{red}
integer\textgreater} ionizationState\argin,\textcolor{red}{\textless double precision\textgreater}
temperature\argin,\textcolor{red}{\textless integer\textgreater} level\argin) Returns
the radiative recombination rate.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the
new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

atomicRecombinationRateRadiativeVerner1996

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names al-
lowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((atomicRecombinationRateRadiativeClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which
could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision rate(\textcolor{red}{\textless integer\textgreater} atomicNumber\argin,\textcolor{red}
integer\textgreater} ionizationState\argin,\textcolor{red}{\textless double precision\textgreater}
temperature\argin,\textcolor{red}{\textless integer\textgreater} level\argin) Returns
the radiative recombination rate.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the
new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

blackHoleBinaryInitialSeparationClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((blackHoleBinaryInitialSeparationClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

double precision separationInitial(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless type((treeNode))\textgreater} nodeHost\arginout) Computes the initial separation of a newly formed black hole binary black holes.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

blackHoleBinaryInitialSeparationSpheroidRadiusFraction

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((blackHoleBinaryInitialSeparationClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`double precision separationInitial(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless type((treeNode))\textgreater} nodeHost\arginout)` Computes the initial separation of a newly formed black hole binary black holes.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

blackHoleBinaryInitialSeparationTidalRadius

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((blackHoleBinaryInitialSeparationClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

double precision separationInitial(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless type((treeNode))\textgreater} nodeHost\arginout) Computes the initial separation of a newly formed black hole binary black holes.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

blackHoleBinaryInitialSeparationVolonteri2003

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((blackHoleBinaryInitialSeparationClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

double precision separationInitial(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless type((treeNode))\textgreater} nodeHost\arginout) Computes the initial separation of a newly formed black hole binary black holes.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

blackHoleBinaryMergerClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((blackHoleBinaryMergerClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

void merge(\textcolor{red}{\textless double precision\textgreater} blackHoleMassA\argin,\textcolor{red} double precision\textgreater} blackHoleMassB\argin,\textcolor{red}{\textless double precision\textgreater} blackHoleSpinA\argin,\textcolor{red}{\textless double precision\textgreater} blackHoleSpinB\argin,\textcolor{red}{\textless double precision\textgreater} blackHoleMassFinal\arg double precision\textgreater} blackHoleSpinFinal\argout) The the properties of the black hole resulting from a binary merger.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

blackHoleBinaryMergerRezzolla2008

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((blackHoleBinaryMergerClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`void merge(\textcolor{red}{\textless double precision\textgreater} blackHoleMassA\argin,\textcolor{red} double precision\textgreater} blackHoleMassB\argin,\textcolor{red}{\textless double precision\textgreater} blackHoleSpinA\argin,\textcolor{red}{\textless double precision\textgreater} blackHoleSpinB\argin,\textcolor{red}{\textless double precision\textgreater} blackHoleMassFinal\arg double precision\textgreater} blackHoleSpinFinal\argout)` The the properties of the black hole resulting from a binary merger.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

blackHoleBinaryRecoilCampanelli2007

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((blackHoleBinaryRecoilClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision velocity(\textcolor{red}{\textless class((nodeComponentBlackHole))\textgreater} blackHole1\arginout,\textcolor{red}{\textless class((nodeComponentBlackHole))\textgreater} blackHole2\arginout)` Computes the recoil velocity of the given pair of merging black holes.

blackHoleBinaryRecoilClass

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((blackHoleBinaryRecoilClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision velocity(\textcolor{red}{\textless class((nodeComponentBlackHole))\textgreater} blackHole1\arginout,\textcolor{red}{\textless class((nodeComponentBlackHole))\textgreater} blackHole2\arginout) Computes the recoil velocity of the given pair of merging black holes.

blackHoleBinaryRecoilZero

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((blackHoleBinaryRecoilClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision velocity(\textcolor{red}{\textless class((nodeComponentBlackHole))\textgreater} blackHole1\arginout,\textcolor{red}{\textless class((nodeComponentBlackHole))\textgreater} blackHole2\arginout) Computes the recoil velocity of the given pair of merging black holes.

blackHoleBinarySeparationGrowthRateClass

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((blackHoleBinarySeparationGrowthRateClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision growthRate(\textcolor{red}{\textless class((nodeComponentBlackHole))\textgreater} blackHole\arginout)` Computes the rate of growth of the separation of the given black hole and its binary companion in units of Mpc/Gyr.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

blackHoleBinarySeparationGrowthRateStandard

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((blackHoleBinarySeparationGrowthRateClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

double precision growthRate(\textcolor{red}{\textless class((nodeComponentBlackHole))\textgreater} blackHole\arginout) Computes the rate of growth of the separation of the given black hole and its binary companion in units of Mpc/Gyr.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

blackHoleBinarySeparationGrowthRateZero

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((blackHoleBinarySeparationGrowthRateClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision growthRate(\textcolor{red}{\textless class((nodeComponentBlackHole))\textgreater} blackHole\arginout) Computes the rate of growth of the separation of the given black hole and its binary companion in units of Mpc/Gyr.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

cap

<logical> pointIncluded(<double(3)> point→) Return true if the given point lives inside the MANGLE cap.

chemicalAbundances

<double> abundance(<integer> moleculeIndex→) Returns the abundance of a chemical given its index.

<void> abundanceSet(<integer> moleculeIndex→, <double> abundance→) Sets the abundance of a chemical given its index.

<type(chemicalAbundances)> add(<type(chemicalAbundances)> abundances2→) Add two chemical abundances.

<void> builder(<*type(node)> chemicalAbundancesDefinition→) Build a chemical abundances object from an XML definition.

<void> deserialize(<double(:)> chemicalAbundancesArray→) Deserialize a chemical abundances object from an array.

<void> destroy() Destroys a chemical abundances object.

<type(chemicalAbundances)> divide(<double> divisor→) Divide a chemical abundance by a scalar.

<void> dump() Dump a chemical abundances object.

<void> dumpRaw(<integer> fileHandle→) Dump a chemical abundances object in binary.

<void> enforcePositive() Enforces all chemical values to be positive.

<void> increment(<type(chemicalAbundances)> addAbundances→) Increment a chemical abundances object.

<logical> isZero() Return true if a chemicals object is zero.

<void> massToNumber(<type(chemicalAbundances)> chemicalsByNumber↔) Converts from abundances by mass to abundances by number.

<type(chemicalAbundances)> multiply(<double> multiplier→) Multiply a chemical abundance by a scalar.

<integer(c_size_t) > nonStaticSizeOf() Returns the size of any non-static components of the type.

<void> numberToMass(<type(chemicalAbundances)> chemicalsByMass↔) Converts from abundances by number to abundances by mass.

<void> readRaw(<integer> fileHandle→) Read a chemical abundances object in binary.

<void> reset() Resets abundances to zero.

<void> serialize(<double(:)> chemicalAbundancesArray←) Serialize a chemical abundances object to an array.

<integer> serializeCount() Return a count of the number of properties in a serialized chemical abundances object.

<void> setToUnity() Set abundances to unity.

<type(chemicalAbundances)> subtract(<type(chemicalAbundances)> abundances2→) Subtract one chemical abundance from another.

chemicalReactionRateClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((chemicalReactionRateClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

void rates(\textcolor{red}{\textless double precision\textgreater} temperature\argin,\textcolor{red}{\t
type((chemicalAbundances))\textgreater} chemicalDensity\argin,\textcolor{red}{\textless
class((radiationFieldClass))\textgreater} radiation\arginout,\textcolor{red}{\textless
type((chemicalAbundances))\textgreater} chemicalRates\arginout,\textcolor{red}{\textless
type((treeNode))\textgreater} node\arginout) Return the collisional excitation cooling rate , in units of $\mathrm{J/m^3/s}$, for ion of given atomicNumber and electronNumber at temperature T (in Kelvin).

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

chemicalReactionRateHydrogenNetwork

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((chemicalReactionRateClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<void> rateH2_Electron_to_2H_Electron(<double> temperature→, <class(radiationFieldClass) radiation>↔, <type(chemicalAbundances) chemicalDensity>→, <type(chemicalAbundances)> chemicalRates↔)` Computes the rate (in units of $\mathrm{c}^{-3}\ \mathrm{s}^{-1}$) for the reaction $\mathrm{H}_2 + \mathrm{e}^- \rightarrow 2\mathrm{H} + \mathrm{e}^-$.

`<void> rateH2_Gamma_to_2H(<double> temperature→, <class(radiationFieldClass) radiation>↔, <type(chemicalAbundances) chemicalDensity>→, <type(chemicalAbundances)> chemicalRates↔, <type(treeNode)> node↔)` Computes the rate (in units of $\mathrm{cm}^{-3}\ \mathrm{s}^{-1}$) for the reaction $\mathrm{H}_2^+ + \gamma \rightarrow 2\mathrm{H}^+ + \mathrm{e}^-$.

`<void> rateH2_Gamma_to_H2plus_Electron(<double> temperature→, <class(radiationFieldClass) radiation>↔, <type(chemicalAbundances) chemicalDensity>→, <type(chemicalAbundances)> chemicalRates↔, <type(treeNode)> node↔)` Computes the rate (in units of $\mathrm{cm}^{-3}\ \mathrm{s}^{-1}$) for the reaction $\mathrm{H}_2 + \gamma \rightarrow \mathrm{H}_2^+ + \mathrm{e}^-$.

`<void> rateH2_Gamma_to_H2star_to_2H(<double> temperature→, <class(radiationFieldClass) radiation>↔, <type(chemicalAbundances) chemicalDensity>→, <type(chemicalAbundances)> chemicalRates↔, <type(treeNode)> node↔)` Computes the rate (in units of $\mathrm{cm}^{-3}\ \mathrm{s}^{-1}$) for the reaction $\mathrm{H}_2^+ + \gamma \rightarrow 2\mathrm{H}^+ + \mathrm{e}^-$.

`<void> rateH2_H_to_3H(<double> temperature→, <class(radiationFieldClass) radiation>↔, <type(chemicalAbundances) chemicalDensity>→, <type(chemicalAbundances)> chemicalRates↔)` Computes the rate (in units of $\mathrm{c}^{-3}\ \mathrm{s}^{-1}$) for the reaction $\mathrm{H}_2 + \mathrm{H} \rightarrow 3\mathrm{H}$.

`<void> rateH2_Hplus_to_H2plus_H(<double> temperature→, <class(radiationFieldClass) radiation>↔, <type(chemicalAbundances) chemicalDensity>→, <type(chemicalAbundances)> chemicalRates↔)` Computes the rate (in units of $\mathrm{c}^{-3}\ \mathrm{s}^{-1}$) for the reaction $\mathrm{H}_2 + \mathrm{H}^+ \rightarrow \mathrm{H}_2^+ + \mathrm{H}$.

`<void> rateH2plus_Electron_to_2H(<double> temperature→, <class(radiationFieldClass) radiation>↔, <type(chemicalAbundances) chemicalDensity>→, <type(chemicalAbundances)> chemicalRates↔)` Computes the rate (in units of $\mathrm{c}^{-3}\ \mathrm{s}^{-1}$) for the reaction $\mathrm{H}_2^+ + \mathrm{e}^- \rightarrow 2\mathrm{H}$.

`<void> rateH2plus_Gamma_to_2Hplus_Electron(<double> temperature→, <class(radiationFieldClass) radiation>↔, <type(chemicalAbundances) chemicalDensity>→, <type(chemicalAbundances)> chemicalRates↔, <type(treeNode)> node↔)` Computes the rate (in units of $\mathrm{cm}^{-3}\ \mathrm{s}^{-1}$) for the reaction $\mathrm{H}_2^+ + \gamma \rightarrow 2\mathrm{H}^+ + \mathrm{e}^-$.

`<void> rateH2plus_Gamma_to_H_Hplus(<double> temperature→, <class(radiationFieldClass) radiation>↔, <type(chemicalAbundances) chemicalDensity>→, <type(chemicalAbundances)> chemicalRates↔, <type(treeNode)> node↔)` Computes the rate (in units of $\mathrm{c}^{-3}\ \mathrm{s}^{-1}$) for the reaction $\mathrm{H}_2^+ + \gamma \rightarrow \mathrm{H} + \mathrm{H}^+$.

`<void> rateH2plus_H_to_H2_Hplus(<double> temperature→, <class(radiationFieldClass) radiation>↔, <type(chemicalAbundances) chemicalDensity>→, <type(chemicalAbundances)> chemicalRates↔)` Computes the rate (in units of $\mathrm{c}^{-3}\ \mathrm{s}^{-1}$) for the reaction $\mathrm{H}_2^+ + \mathrm{H} \rightarrow \mathrm{H}_2 + \mathrm{H}^+$.

`<void> rateH2plus_Hminus_to_H2_H(<double> temperature→, <class(radiationFieldClass) radiation>↔, <type(chemicalAbundances) chemicalDensity>→, <type(chemicalAbundances)> chemicalRates↔)` Computes the rate (in units of $\mathrm{c}^{-3}\ \mathrm{s}^{-1}$) for the reaction $\mathrm{H}_2^+ + \mathrm{H}^- \rightarrow \mathrm{H}_2 + \mathrm{H}$.

`<void> rateH_Electron_to_Hminus_Photon(<double> temperature→, <class(radiationFieldClass) radiation>↔, <type(chemicalAbundances) chemicalDensity>→, <type(chemicalAbundances)> chemicalRates↔)` Computes the rate (in units of $\mathrm{cm}^{-3}\ \mathrm{s}^{-1}$) for the reaction $\mathrm{H} + \mathrm{e}^- \rightarrow \mathrm{H}^- + \gamma$.

`<void> rateH_Electron_to_Hplus_2Electron(<double> temperature→, <class(radiationFieldClass) radiation>↔, <type(chemicalAbundances) chemicalDensity>→, <type(chemicalAbundances)> chemicalRates↔)` Computes the rate (in units of $\mathrm{c}^{-3}\ \mathrm{s}^{-1}$) for the reaction $\mathrm{H} + \mathrm{e}^- \rightarrow \mathrm{H}^+ + 2\mathrm{e}^-$.

`<void> rateH_Gamma_to_Hplus_Electron(<double> temperature→, <class(radiationFieldClass) radiation>↔, <type(chemicalAbundances) chemicalDensity>→, <type(chemicalAbundances)> chemicalRates↔, <type(treeNode)> node↔)` Computes the rate (in units of $\mathrm{cm}^{-3}\ \mathrm{s}^{-1}$) for the reaction $\mathrm{H} + \gamma \rightarrow \mathrm{H}^+ + \mathrm{e}^-$.

`<void> rateH_Hminus_to_H2_Electron(<double> temperature→, <class(radiationFieldClass) radiation>↔, <type(chemicalAbundances) chemicalDensity>→, <type(chemicalAbundances)> chemicalRates↔)` Computes the rate (in units of $\mathrm{cm}^{-3}\ \mathrm{s}^{-1}$) for the reaction $\mathrm{H} + \mathrm{H}^- \rightarrow \mathrm{H}_2 + \mathrm{e}^-$.

`<void> rateH_Hplus_to_H2plus_Photon(<double> temperature→, <class(radiationFieldClass) radiation>↔, <type(chemicalAbundances) chemicalDensity>→, <type(chemicalAbundances)> chemicalRates↔)` Computes the rate (in units of $\mathrm{c}^{-3}\ \mathrm{s}^{-1}$) for the reaction $\mathrm{H} + \mathrm{H}^+ \rightarrow \mathrm{H}_2^+ + \gamma$.

`<void> rateHminus_Electron_to_H_2Electron(<double> temperature→, <class(radiationFieldClass) radiation>↔, <type(chemicalAbundances) chemicalDensity>→, <type(chemicalAbundances)> chemicalRates↔)` Computes the rate (in units of $\mathrm{cm}^{-3}\ \mathrm{s}^{-1}$) for the reaction $\mathrm{H}_2^+ + \mathrm{H} \rightarrow \mathrm{H}_2 + \mathrm{H}^+$.

`<void> rateHminus_Gamma_to_H_Electron(<double> temperature→, <class(radiationFieldClass) radiation>↔, <type(chemicalAbundances) chemicalDensity>→, <type(chemicalAbundances)> chemicalRates↔, <type(treeNode)> node↔)` Computes the rate (in units of $\mathrm{c}^{-3}\ \mathrm{s}^{-1}$) for the reaction $\mathrm{H}^- + \gamma \rightarrow \mathrm{H} + \mathrm{e}^-$.

`<void> rateHminus_H_to_2H_Electron(<double> temperature→, <class(radiationFieldClass) radiation>↔, <type(chemicalAbundances) chemicalDensity>→, <type(chemicalAbundances)> chemicalRates↔)` Computes the rate (in units of $\mathrm{c}^{-3}\ \mathrm{s}^{-1}$) for the reaction $\mathrm{H}^- + \mathrm{H} \rightarrow 2\mathrm{H} + \mathrm{e}^-$.

<void> rateHminus_Hplus_to_2H(<double> temperature→, <class(radiationFieldClass) radiation>↔, <type(chemicalAbundances) chemicalDensity>→, <type(chemicalAbundances)> chemicalRates↔) Computes the rate (in units of c^{-3} s^{-1}) for the reaction $H_2^+ + H \rightarrow H_2 + H^+$.

<void> rateHminus_Hplus_to_H2plus_Electron(<double> temperature→, <class(radiationFieldClass) radiation>↔, <type(chemicalAbundances) chemicalDensity>→, <type(chemicalAbundances)> chemicalRates↔) Computes the rate (in units of cm^{-3} s^{-1}) for the reaction $H^- + H^+ \rightarrow H_2^+ + e^-$.

<void> rateHplus_Electron_to_H_Photon(<double> temperature→, <class(radiationFieldClass) radiation>↔, <type(chemicalAbundances) chemicalDensity>→, <type(chemicalAbundances)> chemicalRates↔) Computes the rate (in units of c^{-3} s^{-1}) for the reaction $H^+ + e^- \rightarrow H + \gamma$.

void rates(\textcolor{red}{\textless double precision\textgreater} temperature\argin,\textcolor{red}{\t type((chemicalAbundances))\textgreater} chemicalDensity\argin,\textcolor{red}{\textless class((radiationFieldClass))\textgreater} radiation\arginout,\textcolor{red}{\textless type((chemicalAbundances))\textgreater} chemicalRates\arginout,\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Return the collisional excitation cooling rate , in units of $J/m^3/s$, for ion of given atomicNumber and electronNumber at temperature T (in Kelvin).

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

chemicalReactionRateZero

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((chemicalReactionRateClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

void rates(\textcolor{red}{\textless double precision\textgreater} temperature\argin,\textcolor{red}{\t
type((chemicalAbundances))\textgreater} chemicalDensity\argin,\textcolor{red}{\textless
class((radiationFieldClass))\textgreater} radiation\arginout,\textcolor{red}{\textless
type((chemicalAbundances))\textgreater} chemicalRates\arginout,\textcolor{red}{\textless
type((treeNode))\textgreater} node\arginout) Return the collisional excitation cooling rate , in units of J/m^3/s, for ion of given atomicNumber and electronNumber at temperature T (in Kelvin).

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

chemicalStateAtomicCIECloudy

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void chemicalDensities(\textcolor{red}{\textless type((chemicalAbundances))\textgreater}
chemicalDensities\arginout,\textcolor{red}{\textless double precision\textgreater}
numberDensityHydrogen\argin,\textcolor{red}{\textless double precision\textgreater}
temperature\argin,\textcolor{red}{\textless type((abundances))\textgreater} gasAbundances\argin,\t
class((radiationFieldClass))\textgreater} radiation\arginout) Return the densities of chemical species at the given temperature and hydrogen density for the specified set of abundances and radiation field. Units of the returned electron density are cm$^-$3.

void deepCopy(\textcolor{red}{\textless class((chemicalStateClass))\textgreater} destination\arginout)
Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision electronDensity(\textcolor{red}{\textless double precision\textgreater}
numberDensityHydrogen\argin,\textcolor{red}{\textless double precision\textgreater}
temperature\argin,\textcolor{red}{\textless type((abundances))\textgreater} gasAbundances\argin,\t
class((radiationFieldClass))\textgreater} radiation\arginout) Return the electron density at the given temperature and hydrogen density for the specified set of abundances and radiation field. Units of the returned electron density are cm$^-$3.

double precision electronDensityDensityLogSlope(\textcolor{red}{\textless double precision\textgreater} numberDensityHydrogen\argin,\textcolor{red}{\textless double precision\textgreater} temperature\argin,\textcolor{red}{\textless type((abundances))\textgreater} gasAbundances\argin,\t class((radiationFieldClass))\textgreater} radiation\arginout) Return the logarithmic gradient of electron density with respect to density at the given temperature and hydrogen density for the specified set of abundances and radiation field.

double precision electronDensityTemperatureLogSlope(\textcolor{red}{\textless double precision\textgrea numberDensityHydrogen\argin,\textcolor{red}{\textless double precision\textgreater} temperature\argin,\textcolor{red}{\textless type((abundances))\textgreater} gasAbundances\argin,\t class((radiationFieldClass))\textgreater} radiation\arginout) Return the logarithmic gradient of electron density with temperature at the given temperature and hydrogen density for the specified set of abundances and radiation field.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<double> interpolate(<integer(c_size_t)> iTemperature→, <double> hTemperature→, <integer(c_size_t)> iMetallicity→, <double> hMetallicity→, <double(:,:)> density→) Interpolate in the given density table.

void interpolatingFactors(<double> temperature→, <double> metallicity→, <integer(c_size_t)> iTemperature←, <double> hTemperature←, <integer(c_size_t)> iMetallicity←, <double> hMetallicity←) Compute interpolating factors in a CIE chemical state file.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

void readFile(<char(len=*)> fileName→) Read the named chemical state file.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

void tabulate(<type(abundances)> gasAbunances→) Run Cloudy to tabulate chemical state as necessary.

chemicalStateCIEFile

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void chemicalDensities(\textcolor{red}{\textless type((chemicalAbundances))\textgreater} chemicalDensities\arginout,\textcolor{red}{\textless double precision\textgreater} numberDensityHydrogen\argin,\textcolor{red}{\textless double precision\textgreater} temperature\argin,\textcolor{red}{\textless type((abundances))\textgreater} gasAbundances\argin,\t class((radiationFieldClass))\textgreater} radiation\arginout)` Return the densities of chemical species at the given temperature and hydrogen density for the specified set of abundances and radiation field. Units of the returned electron density are cm^{-}3.

`void deepCopy(\textcolor{red}{\textless class((chemicalStateClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision electronDensity(\textcolor{red}{\textless double precision\textgreater} numberDensityHydrogen\argin,\textcolor{red}{\textless double precision\textgreater} temperature\argin,\textcolor{red}{\textless type((abundances))\textgreater} gasAbundances\argin,\t class((radiationFieldClass))\textgreater} radiation\arginout)` Return the electron density at the given temperature and hydrogen density for the specified set of abundances and radiation field. Units of the returned electron density are cm^{-}3.

`double precision electronDensityDensityLogSlope(\textcolor{red}{\textless double precision\textgreater} numberDensityHydrogen\argin,\textcolor{red}{\textless double precision\textgreater} temperature\argin,\textcolor{red}{\textless type((abundances))\textgreater} gasAbundances\argin,\t class((radiationFieldClass))\textgreater} radiation\arginout)` Return the logarithmic gradient of electron density with respect to density at the given temperature and hydrogen density for the specified set of abundances and radiation field.

`double precision electronDensityTemperatureLogSlope(\textcolor{red}{\textless double precision\textgrea numberDensityHydrogen\argin,\textcolor{red}{\textless double precision\textgreater} temperature\argin,\textcolor{red}{\textless type((abundances))\textgreater} gasAbundances\argin,\t class((radiationFieldClass))\textgreater} radiation\arginout)` Return the logarithmic gradient of electron density with temperature at the given temperature and hydrogen density for the specified set of abundances and radiation field.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<double> interpolate(<integer(c_size_t)> iTemperature→, <double> hTemperature→, <integer(c_size_t)> iMetallicity→, <double> hMetallicity→, <double(:,:)> density→)` Interpolate in the given density table.

`void interpolatingFactors(<double> temperature→, <double> metallicity→, <integer(c_size_t)> iTemperature←, <double> hTemperature←, <integer(c_size_t)> iMetallicity←, <double> hMetallicity←)` Compute interpolating factors in a CIE chemical state file.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`void readFile(<char(len=*)> fileName→)` Read the named chemical state file.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

chemicalStateClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void chemicalDensities(\textcolor{red}{\textless type((chemicalAbundances))\textgreater} chemicalDensities\arginout,\textcolor{red}{\textless double precision\textgreater} numberDensityHydrogen\argin,\textcolor{red}{\textless double precision\textgreater} temperature\argin,\textcolor{red}{\textless type((abundances))\textgreater} gasAbundances\argin,\t class((radiationFieldClass))\textgreater} radiation\arginout) Return the densities of chemical species at the given temperature and hydrogen density for the specified set of abundances and radiation field. Units of the returned electron density are cm^{-}3.

void deepCopy(\textcolor{red}{\textless class((chemicalStateClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision electronDensity(\textcolor{red}{\textless double precision\textgreater} numberDensityHydrogen\argin,\textcolor{red}{\textless double precision\textgreater} temperature\argin,\textcolor{red}{\textless type((abundances))\textgreater} gasAbundances\argin,\t class((radiationFieldClass))\textgreater} radiation\arginout) Return the electron density at the given temperature and hydrogen density for the specified set of abundances and radiation field. Units of the returned electron density are cm^{-}3.

double precision electronDensityDensityLogSlope(\textcolor{red}{\textless double precision\textgreater} numberDensityHydrogen\argin,\textcolor{red}{\textless double precision\textgreater} temperature\argin,\textcolor{red}{\textless type((abundances))\textgreater} gasAbundances\argin,\t class((radiationFieldClass))\textgreater} radiation\arginout) Return the logarithmic gradient of electron density with respect to density at the given temperature and hydrogen density for the specified set of abundances and radiation field.

`double precision electronDensityTemperatureLogSlope(\textcolor{red}{\textless double precision\textgrea numberDensityHydrogen\argin,\textcolor{red}{\textless double precision\textgreater} temperature\argin,\textcolor{red}{\textless type((abundances))\textgreater} gasAbundances\argin,\t class((radiationFieldClass))\textgreater} radiation\arginout)` Return the logarithmic gradient of electron density with temperature at the given temperature and hydrogen density for the specified set of abundances and radiation field.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

chemicalStructure

`<integer> charge()` Return the charge of a chemical.

`<void> export(<character(len=*)> coutputFile→)` Write a chemical structure to a CML file.

`<double> mass()` Return the mass of a chemical in atomic mass units.

`<void> retrieve(<character(len=*)> chemicalName→)` Get a chemical from the database.

coldModeInfallRateClass

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((coldModeInfallRateClass))\textgreater} destination\argin` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`double precision infallRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Returns the cold mode infall rate for `thisNode` (in units of $M_\odot$ Gyr^{-1}).

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

coldModeInfallRateDynamicalTime

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((coldModeInfallRateClass))\textgreater} destination\argir` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`double precision infallRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Returns the cold mode infall rate for `thisNode` (in units of $M_\odot$ Gyr^{-1}).

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

conditionalMassFunctionBehroozi2010

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`<void> compute(<double> massHalo→, <double> mass→, <double> numberCentrals←, <double> numberSatellites←)` Compute the cumulative conditional mass function, $\langle N(M_\star|M_{\rm halo})\rangle \equiv \phi(M_\star|M_{\rm halo})$.

`void deepCopy(\textcolor{red}{\textless class((conditionalMassFunctionClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision massFunction(\textcolor{red}{\textless double precision\textgreater} massHalo\argin,\t double precision\textgreater} mass\argin,\textcolor{red}{\textless integer\textgreater} galaxyType\argin)` Return the cumulative conditional mass function, $\langle N(M_\star|M_{\rm halo})\rangle \equiv \phi(M_\star|M_{\rm halo})$.

`double precision massFunctionVariance(\textcolor{red}{\textless double precision\textgreater} massHalo\argin,\textcolor{red}{\textless double precision\textgreater} massLow\argin,\textcolor{red double precision\textgreater} massHigh\argin)` Return the variance in the cumulative conditional mass function, $\langle N(M_\star|M_{\rm halo})\rangle \equiv \phi(M_\star|M_{\rm halo})$.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

conditionalMassFunctionClass

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((conditionalMassFunctionClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision massFunction(\textcolor{red}{\textless double precision\textgreater} massHalo\argin,\t double precision\textgreater} mass\argin,\textcolor{red}{\textless integer\textgreater} galaxyType\argin)` Return the cumulative conditional mass function, $\langle N(M_\star|M_{\rm halo})\rangle \equiv \phi(M_\star|M_{\rm halo})$.

`double precision massFunctionVariance(\textcolor{red}{\textless double precision\textgreater} massHalo\argin,\textcolor{red}{\textless double precision\textgreater} massLow\argin,\textcolor{red double precision\textgreater} massHigh\argin)` Return the variance in the cumulative conditional mass function, $\langle N(M_\star|M_{\rm halo})\rangle \equiv \phi(M_\star|M_{\rm halo})$.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

coolingFunctionAtomicCIECloudy

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

double precision coolingFunction(\textcolor{red}{\textless double precision\textgreater} numberDensityHydrogen\argin,\textcolor{red}{\textless double precision\textgreater} temperature\argin,\textcolor{red}{\textless type((abundances))\textgreater} gasAbundances\argin,\t type((chemicalAbundances))\textgreater} chemicalDensities\argin,\textcolor{red}{\textless class((radiationFieldClass))\textgreater} radiation\arginout) Return the cooling function at the given temperature and hydrogen density for the specified set of abundances and radiation field. Units of the returned cooling function are the traditional ergs cm$^-$3 s^{-1}.

double precision coolingFunctionDensityLogSlope(\textcolor{red}{\textless double precision\textgreater} numberDensityHydrogen\argin,\textcolor{red}{\textless double precision\textgreater} temperature\argin,\textcolor{red}{\textless type((abundances))\textgreater} gasAbundances\argin,\t type((chemicalAbundances))\textgreater} chemicalDensities\argin,\textcolor{red}{\textless class((radiationFieldClass))\textgreater} radiation\arginout) Return $\mathrm{d}\ln\Lambda/\mathrm{d}\ln\rho$ for a cooling function at the given temperature and hydrogen density for the specified set of abundances and radiation field.

double precision coolingFunctionTemperatureLogSlope(\textcolor{red}{\textless double precision\textgrea numberDensityHydrogen\argin,\textcolor{red}{\textless double precision\textgreater} temperature\argin,\textcolor{red}{\textless type((abundances))\textgreater} gasAbundances\argin,\t type((chemicalAbundances))\textgreater} chemicalDensities\argin,\textcolor{red}{\textless class((radiationFieldClass))\textgreater} radiation\arginout) Return $\mathrm{d}\ln\Lambda/\mathrm{d}\ln T$ for a cooling function at the given temperature and hydrogen density for the specified set of abundances and radiation field.

void deepCopy(\textcolor{red}{\textless class((coolingFunctionClass))\textgreater} destination\arginout Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<double> interpolate(<integer(c_size_t)> iTemperature→, <double> hTemperature→, <integer(c_size_t)> iMetallicity→, <double> hMetallicity→) Interpolate in the cooling function.

void interpolatingFactors(<double> temperature→, <double> metallicity→, <integer(c_size_t)> iTemperature←, <double> hTemperature←, <integer(c_size_t)> iMetallicity←, <double> hMetallicity←) Compute interpolating factors in a CIE cooling function file.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

void readFile(<char(len=*)> fileName→) Read the named cooling function file.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`void tabulate(<type(abundances)> gasAbunances→)` Run CLOUDY to tabulate the cooling function as necessary.

coolingFunctionCIEFile

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`double precision coolingFunction(\textcolor{red}{\textless double precision\textgreater} numberDensityHydrogen\argin,\textcolor{red}{\textless double precision\textgreater} temperature\argin,\textcolor{red}{\textless type((abundances))\textgreater} gasAbundances\argin,\t type((chemicalAbundances))\textgreater} chemicalDensities\argin,\textcolor{red}{\textless class((radiationFieldClass))\textgreater} radiation\arginout)` Return the cooling function at the given temperature and hydrogen density for the specified set of abundances and radiation field. Units of the returned cooling function are the traditional ergs $\mathrm{cm}^{-}3\ \mathrm{s}^{-1}$.

`double precision coolingFunctionDensityLogSlope(\textcolor{red}{\textless double precision\textgreater} numberDensityHydrogen\argin,\textcolor{red}{\textless double precision\textgreater} temperature\argin,\textcolor{red}{\textless type((abundances))\textgreater} gasAbundances\argin,\t type((chemicalAbundances))\textgreater} chemicalDensities\argin,\textcolor{red}{\textless class((radiationFieldClass))\textgreater} radiation\arginout)` Return $\mathrm{d}\ln\Lambda/\mathrm{d}\ln\rho$ for a cooling function at the given temperature and hydrogen density for the specified set of abundances and radiation field.

`double precision coolingFunctionTemperatureLogSlope(\textcolor{red}{\textless double precision\textgrea numberDensityHydrogen\argin,\textcolor{red}{\textless double precision\textgreater} temperature\argin,\textcolor{red}{\textless type((abundances))\textgreater} gasAbundances\argin,\t type((chemicalAbundances))\textgreater} chemicalDensities\argin,\textcolor{red}{\textless class((radiationFieldClass))\textgreater} radiation\arginout)` Return $\mathrm{d}\ln\Lambda/\mathrm{d}\ln T$ for a cooling function at the given temperature and hydrogen density for the specified set of abundances and radiation field.

`void deepCopy(\textcolor{red}{\textless class((coolingFunctionClass))\textgreater} destination\arginout` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<double> interpolate(<integer(c_size_t)> iTemperature→, <double> hTemperature→, <integer(c_size_t)> iMetallicity→, <double> hMetallicity→) Interpolate in the cooling function.

void interpolatingFactors(<double> temperature→, <double> metallicity→, <integer(c_size_t)> iTemperature←, <double> hTemperature←, <integer(c_size_t)> iMetallicity←, <double> hMetallicity←) Compute interpolating factors in a CIE cooling function file.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

void readFile(<char(len=*)> fileName→) Read the named cooling function file.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

coolingFunctionClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

double precision coolingFunction(\textcolor{red}{\textless double precision\textgreater} numberDensityHydrogen\argin,\textcolor{red}{\textless double precision\textgreater} temperature\argin,\textcolor{red}{\textless type((abundances))\textgreater} gasAbundances\argin,\t type((chemicalAbundances))\textgreater} chemicalDensities\argin,\textcolor{red}{\textless class((radiationFieldClass))\textgreater} radiation\arginout) Return the cooling function at the given temperature and hydrogen density for the specified set of abundances and radiation field. Units of the returned cooling function are the traditional ergs cm^{-}3 s^{-1}.

double precision coolingFunctionDensityLogSlope(\textcolor{red}{\textless double precision\textgreater} numberDensityHydrogen\argin,\textcolor{red}{\textless double precision\textgreater} temperature\argin,\textcolor{red}{\textless type((abundances))\textgreater} gasAbundances\argin,\t type((chemicalAbundances))\textgreater} chemicalDensities\argin,\textcolor{red}{\textless class((radiationFieldClass))\textgreater} radiation\arginout) Return $\mathrm{d}\ln\Lambda/\mathrm{d}\ln\rho$ for a cooling function at the given temperature and hydrogen density for the specified set of abundances and radiation field.

`double precision coolingFunctionTemperatureLogSlope(\textcolor{red}{\textless double precision\textgrea numberDensityHydrogen\argin,\textcolor{red}{\textless double precision\textgreater} temperature\argin,\textcolor{red}{\textless type((abundances))\textgreater} gasAbundances\argin,\t type((chemicalAbundances))\textgreater} chemicalDensities\argin,\textcolor{red}{\textless class((radiationFieldClass))\textgreater} radiation\arginout)` Return $\mathrm{d}\ln\Lambda/\mathrm{d}\ln T$ for a cooling function at the given temperature and hydrogen density for the specified set of abundances and radiation field.

`void deepCopy(\textcolor{red}{\textless class((coolingFunctionClass))\textgreater} destination\arginout` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

coolingFunctionCMBCompton

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`double precision coolingFunction(\textcolor{red}{\textless double precision\textgreater} numberDensityHydrogen\argin,\textcolor{red}{\textless double precision\textgreater} temperature\argin,\textcolor{red}{\textless type((abundances))\textgreater} gasAbundances\argin,\t type((chemicalAbundances))\textgreater} chemicalDensities\argin,\textcolor{red}{\textless class((radiationFieldClass))\textgreater} radiation\arginout)` Return the cooling function at the given temperature and hydrogen density for the specified set of abundances and radiation field. Units of the returned cooling function are the traditional ergs $\mathrm{cm}^{-}3\ \mathrm{s}^{-1}$.

`double precision coolingFunctionDensityLogSlope(\textcolor{red}{\textless double precision\textgreater} numberDensityHydrogen\argin,\textcolor{red}{\textless double precision\textgreater} temperature\argin,\textcolor{red}{\textless type((abundances))\textgreater} gasAbundances\argin,\textcolor{red}{\textless type((chemicalAbundances))\textgreater} chemicalDensities\argin,\textcolor{red}{\textless class((radiationFieldClass))\textgreater} radiation\arginout)` Return $\mathrm{d}\ln\Lambda/\mathrm{d}\ln\rho$ for a cooling function at the given temperature and hydrogen density for the specified set of abundances and radiation field.

`double precision coolingFunctionTemperatureLogSlope(\textcolor{red}{\textless double precision\textgreater} numberDensityHydrogen\argin,\textcolor{red}{\textless double precision\textgreater} temperature\argin,\textcolor{red}{\textless type((abundances))\textgreater} gasAbundances\argin,\textcolor{red}{\textless type((chemicalAbundances))\textgreater} chemicalDensities\argin,\textcolor{red}{\textless class((radiationFieldClass))\textgreater} radiation\arginout)` Return $\mathrm{d}\ln\Lambda/\mathrm{d}\ln T$ for a cooling function at the given temperature and hydrogen density for the specified set of abundances and radiation field.

`void deepCopy(\textcolor{red}{\textless class((coolingFunctionClass))\textgreater} destination\arginout` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\textcolor{red}{\textless logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

coolingFunctionMolecularHydrogenGalliPalla

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

void commonFactors(<double> numberDensityHydrogen→, <double> temperature→, <double> numberDensityCrit
<double> coolingFunctionLocalThermodynamicEquilibrium←, <double> coolingFunctionLowDensityLimit←)
Compute common factors.

double precision coolingFunction(\textcolor{red}{\textless double precision\textgreater}
numberDensityHydrogen\argin,\textcolor{red}{\textless double precision\textgreater}
temperature\argin,\textcolor{red}{\textless type((abundances))\textgreater} gasAbundances\argin,\t
type((chemicalAbundances))\textgreater} chemicalDensities\argin,\textcolor{red}{\textless
class((radiationFieldClass))\textgreater} radiation\arginout) Return the cooling function at the given temperature and hydrogen density for the specified set of abundances and radiation field. Units of the returned cooling function are the traditional ergs cm^{-}3 s^{-1}.

double precision coolingFunctionDensityLogSlope(\textcolor{red}{\textless double precision\textgreater}
numberDensityHydrogen\argin,\textcolor{red}{\textless double precision\textgreater}
temperature\argin,\textcolor{red}{\textless type((abundances))\textgreater} gasAbundances\argin,\t
type((chemicalAbundances))\textgreater} chemicalDensities\argin,\textcolor{red}{\textless
class((radiationFieldClass))\textgreater} radiation\arginout) Return $\mathrm{d}\ln\Lambda/\mathrm{d}\ln\rho$ for a cooling function at the given temperature and hydrogen density for the specified set of abundances and radiation field.

<double> coolingFunctionH2Plus_Electron(<double> temperature→,<type(chemicalAbundances)>
chemicalDensities→) Compute the cooling function due to H_2^+–e^-.

<double> coolingFunctionH_H2(<double> numberDensityHydrogen→, <double> temperature→,<type(chemicalAbu
chemicalDensities→) Compute the cooling function due to H–H_2.

<double> coolingFunctionH_H2Plus(<double> temperature→,<type(chemicalAbundances)> chemicalDensities→)
Compute the cooling function due to H–H_2^+.

double precision coolingFunctionTemperatureLogSlope(\textcolor{red}{\textless double precision\textgrea
numberDensityHydrogen\argin,\textcolor{red}{\textless double precision\textgreater}
temperature\argin,\textcolor{red}{\textless type((abundances))\textgreater} gasAbundances\argin,\t
type((chemicalAbundances))\textgreater} chemicalDensities\argin,\textcolor{red}{\textless
class((radiationFieldClass))\textgreater} radiation\arginout) Return $\mathrm{d}\ln\Lambda/\mathrm{d}\ln T$ for a cooling function at the given temperature and hydrogen density for the specified set of abundances and radiation field.

void deepCopy(\textcolor{red}{\textless class((coolingFunctionClass))\textgreater} destination\arginout
Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

coolingFunctionSummation

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

double precision coolingFunction(\textcolor{red}{\textless double precision\textgreater} numberDensityHydrogen\argin,\textcolor{red}{\textless double precision\textgreater} temperature\argin,\textcolor{red}{\textless type((abundances))\textgreater} gasAbundances\argin,\t type((chemicalAbundances))\textgreater} chemicalDensities\argin,\textcolor{red}{\textless class((radiationFieldClass))\textgreater} radiation\arginout) Return the cooling function at the given temperature and hydrogen density for the specified set of abundances and radiation field. Units of the returned cooling function are the traditional ergs cm^{-}3 s^{-1}.

double precision coolingFunctionDensityLogSlope(\textcolor{red}{\textless double precision\textgreater} numberDensityHydrogen\argin,\textcolor{red}{\textless double precision\textgreater} temperature\argin,\textcolor{red}{\textless type((abundances))\textgreater} gasAbundances\argin,\t type((chemicalAbundances))\textgreater} chemicalDensities\argin,\textcolor{red}{\textless class((radiationFieldClass))\textgreater} radiation\arginout) Return $\mathrm{d}\ln\Lambda/\mathrm{d}\ln\rho$ for a cooling function at the given temperature and hydrogen density for the specified set of abundances and radiation field.

double precision coolingFunctionTemperatureLogSlope(\textcolor{red}{\textless double precision\textgrea numberDensityHydrogen\argin,\textcolor{red}{\textless double precision\textgreater} temperature\argin,\textcolor{red}{\textless type((abundances))\textgreater} gasAbundances\argin,\t type((chemicalAbundances))\textgreater} chemicalDensities\argin,\textcolor{red}{\textless class((radiationFieldClass))\textgreater} radiation\arginout) Return $\mathrm{d}\ln\Lambda/\mathrm{d}\ln T$ for a cooling function at the given temperature and hydrogen density for the specified set of abundances and radiation field.

void deepCopy(\textcolor{red}{\textless class((coolingFunctionClass))\textgreater} destination\arginout Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

coolingInfallRadiusClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((coolingInfallRadiusClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision radius(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Return the infall radius for node (in units of Mpc).

double precision radiusIncreaseRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Return the rate at which the infall radius grows for node (in units of Mpc/Gyr).

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

coolingInfallRadiusCoolingFreefall

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((coolingInfallRadiusClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`double precision radius(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Return the infall radius for `node` (in units of Mpc).

`double precision radiusIncreaseRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Return the rate at which the infall radius grows for `node` (in units of Mpc/Gyr).

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

coolingInfallRadiusCoolingRadius

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((coolingInfallRadiusClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`double precision radius(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Return the infall radius for `node` (in units of Mpc).

`double precision radiusIncreaseRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Return the rate at which the infall radius grows for `node` (in units of Mpc/Gyr).

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

coolingRadiusBetaProfile

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`<void> calculationReset(<type(table)> node↔)` Reset memoized calculations.

`void deepCopy(\textcolor{red}{\textless class((coolingRadiusClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision radius(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the cooling radius for gas in the hot atmosphere surrounding the galaxy in node in units of Mpc.

double precision radiusGrowthRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the rate of increase of the cooling radius for gas in the hot atmosphere surrounding the galaxy in node in units of Mpc/Gyr.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

coolingRadiusClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((coolingRadiusClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision radius(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the cooling radius for gas in the hot atmosphere surrounding the galaxy in node in units of Mpc.

double precision radiusGrowthRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the rate of increase of the cooling radius for gas in the hot atmosphere surrounding the galaxy in node in units of Mpc/Gyr.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

coolingRadiusIsothermal

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

<void> calculationReset(<type(table)> node↔) Reset memoized calculations.

void deepCopy(\textcolor{red}{\textless class((coolingRadiusClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision radius(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the cooling radius for gas in the hot atmosphere surrounding the galaxy in node in units of Mpc.

double precision radiusGrowthRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the rate of increase of the cooling radius for gas in the hot atmosphere surrounding the galaxy in node in units of Mpc/Gyr.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

coolingRadiusSimple

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

<void> calculationReset(<type(table)> node↔) Reset memoized calculations.

void deepCopy(\textcolor{red}{\textless class((coolingRadiusClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision radius(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the cooling radius for gas in the hot atmosphere surrounding the galaxy in node in units of Mpc.

double precision radiusGrowthRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the rate of increase of the cooling radius for gas in the hot atmosphere surrounding the galaxy in node in units of Mpc/Gyr.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

coolingRateClass

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((coolingRateClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`double precision rate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Returns the cooling rate of gas in the hot atmosphere surrounding the galaxy in `node` in units of $M_\odot$/Gyr.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

coolingRateCole2000

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((coolingRateClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision rate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)
Returns the cooling rate of gas in the hot atmosphere surrounding the galaxy in node in units of $M_\odot$/Gyr.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

coolingRateCutOff

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((coolingRateClass))\textgreater} destination\arginout)
Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision rate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)
Returns the cooling rate of gas in the hot atmosphere surrounding the galaxy in node in units of $M_\odot$/Gyr.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

coolingRateMultiplier

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((coolingRateClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision rate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the cooling rate of gas in the hot atmosphere surrounding the galaxy in node in units of $M_\odot$/Gyr.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

coolingRateNoCoolingSatellites

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((coolingRateClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision rate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the cooling rate of gas in the hot atmosphere surrounding the galaxy in node in units of $M_\odot$/Gyr.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

coolingRateSimple

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((coolingRateClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision rate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)
Returns the cooling rate of gas in the hot atmosphere surrounding the galaxy in node in units of $M_\odot$/Gyr.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

coolingRateSimpleScaling

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((coolingRateClass))\textgreater} destination\arginout)
Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision rate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)
Returns the cooling rate of gas in the hot atmosphere surrounding the galaxy in node in units of $M_\odot$/Gyr.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

coolingRateVelocityMaximumScaling

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((coolingRateClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision rate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the cooling rate of gas in the hot atmosphere surrounding the galaxy in node in units of $M_\odot$/Gyr.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

coolingRateWhiteFrenk1991

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((coolingRateClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`double precision rate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Returns the cooling rate of gas in the hot atmosphere surrounding the galaxy in `node` in units of $M_\odot$/Gyr.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

coolingRateZero

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((coolingRateClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision rate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)
Returns the cooling rate of gas in the hot atmosphere surrounding the galaxy in node in units of $M_\odot$/Gyr.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

coolingSpecificAngularMomentumClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

double precision angularMomentumSpecific(\textcolor{red}{\textless type((treeNode))\textgreater}
node\arginout,\textcolor{red}{\textless double precision\textgreater} radius\argin)
Return the specific angular momentum (in units of km/s Mpc) of cooling gas in node.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((coolingSpecificAngularMomentumClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

coolingSpecificAngularMomentumConstantRotation

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

double precision angularMomentumSpecific(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} radius\argin) Return the specific angular momentum (in units of km/s Mpc) of cooling gas in node.

void autoHook() Insert any event hooks required by this object.

<void> calculationReset(<type(table)> node↔) Reset memoized calculations.

void deepCopy(\textcolor{red}{\textless class((coolingSpecificAngularMomentumClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

coolingSpecificAngularMomentumMean

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`double precision angularMomentumSpecific(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} radius\argin)` Return the specific angular momentum (in units of km/s Mpc) of cooling gas in `node`.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((coolingSpecificAngularMomentumClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

coolingTimeAvailableClass

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((coolingTimeAvailableClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision timeAvailable(\textcolor{red}{\textless type((treeNode))\textgreater}
node\arginout) Return the time available for cooling in node in units of Gyr.

double precision timeAvailableIncreaseRate(\textcolor{red}{\textless type((treeNode))\textgreater}
node\arginout) Return the rate at which the time available for cooling increases in node (dimensionless).

coolingTimeAvailableFormationTime

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((coolingTimeAvailableClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision timeAvailable(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Return the time available for cooling in node in units of Gyr.

double precision timeAvailableIncreaseRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Return the rate at which the time available for cooling increases in node (dimensionless).

coolingTimeAvailableWhiteFrenk1991

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((coolingTimeAvailableClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision timeAvailable(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Return the time available for cooling in node in units of Gyr.

double precision timeAvailableIncreaseRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Return the rate at which the time available for cooling increases in node (dimensionless).

coolingTimeClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((coolingTimeClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision gradientDensityLogarithmic(\textcolor{red}{\textless double precision\textgreater} temperature\argin,\textcolor{red}{\textless double precision\textgreater} density\argin,\textcolor{ type((abundances))\textgreater} gasAbundances \argin,\textcolor{red}{\textless type((chemicalAbund))\textgreater} chemicalDensities \argin,\textcolor{red}{\textless class((radiationFieldClass))\tex radiation \arginout) Returns the logarithmic derivative of cooling time with respect to density for gas in the hot atmosphere surrounding the galaxy.

double precision gradientTemperatureLogarithmic(\textcolor{red}{\textless double precision\textgreater} temperature\argin,\textcolor{red}{\textless double precision\textgreater} density\argin,\textcolor{ type((abundances))\textgreater} gasAbundances \argin,\textcolor{red}{\textless type((chemicalAbund))\textgreater} chemicalDensities \argin,\textcolor{red}{\textless class((radiationFieldClass))\tex radiation \arginout) Returns the logarithmic derivative of cooling time with respect to temperature for gas in the hot atmosphere surrounding the galaxy.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision time(\textcolor{red}{\textless double precision\textgreater} temperature\argin,\textcolor{red}{\textless double precision\textgreater} density\argin,\textcolor{red}{\textless type((abundances))\textgreater} gasAbundances \argin,\textcolor{red}{\textless type((chemicalAbundances))\textgreater} chemicalDensities \argin,\textcolor{red}{\textless class((radiationFieldClass))\textgreater} radiation \arginout) Returns the cooling time for gas in the hot atmosphere surrounding the galaxy in units of Gyr.

coolingTimeSimple

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\textcolor{red}{\textless character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((coolingTimeClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\textcolor{red}{\textless logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision gradientDensityLogarithmic(\textcolor{red}{\textless double precision\textgreater} temperature\argin,\textcolor{red}{\textless double precision\textgreater} density\argin,\textcolor{red}{\textless type((abundances))\textgreater} gasAbundances \argin,\textcolor{red}{\textless type((chemicalAbundances))\textgreater} chemicalDensities \argin,\textcolor{red}{\textless class((radiationFieldClass))\textgreater} radiation \arginout) Returns the logarithmic derivative of cooling time with respect to density for gas in the hot atmosphere surrounding the galaxy.

double precision gradientTemperatureLogarithmic(\textcolor{red}{\textless double precision\textgreater} temperature\argin,\textcolor{red}{\textless double precision\textgreater} density\argin,\textcolor{red}{\textless type((abundances))\textgreater} gasAbundances \argin,\textcolor{red}{\textless type((chemicalAbundances))\textgreater} chemicalDensities \argin,\textcolor{red}{\textless class((radiationFieldClass))\textgreater} radiation \arginout) Returns the logarithmic derivative of cooling time with respect to temperature for gas in the hot atmosphere surrounding the galaxy.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDigest) Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision time(\textcolor{red}{\textless double precision\textgreater} temperature\argin,\textcc double precision\textgreater} density\argin,\textcolor{red}{\textless type((abundances))\textgreater} gasAbundances \argin,\textcolor{red}{\textless type((chemicalAbundances))\textgreater} chemicalDensities \argin,\textcolor{red}{\textless class((radiationFieldClass))\tex radiation \arginout) Returns the cooling time for gas in the hot atmosphere surrounding the galaxy in units of Gyr.

coordinate

<void> fromCartesian(<double(3)> x→) Set the coordinates from a Cartesian system specified as a 3-element array.

<double> rCylindrical() Return the cylindrical radial coordinate.

<double(3)> toCartesian() Return the coordinates in a Cartesian system as a 3-element array.

coordinateCartesian

<void> fromCartesian(<double(3)> x→) Set the coordinates from a Cartesian system specified as a 3-element array.

<double> rCylindrical() Return the cylindrical radial coordinate.

<double(3)> toCartesian() Return the coordinates in a Cartesian system as a 3-element array.

<double> x() Get the x-coordinate.

<void> xSet(<double> x→) set the x-coordinate.

<double> y() Get the y-coordinate.

<void> ySet(<double> y→) set the y-coordinate.

<double> z() Get the z-coordinate.

<void> zSet(<double> z→) set the z-coordinate.

coordinateCylindrical

<void> fromCartesian(<double(3)> x→) Set the coordinates from a Cartesian system specified as a 3-element array.

<double> phi() Get the ϕ-coordinate.

<void> phiSet(<double> phi→) set the ϕ-coordinate.

<double> r() Get the r-coordinate.

<double> rCylindrical() Return the cylindrical radial coordinate.

<void> rSet(<double> r→) set the r-coordinate.

<double(3)> toCartesian() Return the coordinates in a Cartesian system as a 3-element array.

<double> z() Get the z-coordinate.

<void> zSet(<double> z→) set the z-coordinate.

coordinateSpherical

<void> fromCartesian(<double(3)> x→) Set the coordinates from a Cartesian system specified as a 3-element array.

<double> phi() Get the ϕ-coordinate.

<void> phiSet(<double> phi→) set the ϕ-coordinate.

<double> r() Get the r-coordinate.

<double> rCylindrical() Return the cylindrical radial coordinate.

<void> rSet(<double> r→) set the r-coordinate.

<double> theta() Get the θ-coordinate.

<void> thetaSet(<double> theta→) set the θ-coordinate.

<double(3)> toCartesian() Return the coordinates in a Cartesian system as a 3-element array.

cosmologicalMassVarianceClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((cosmologicalMassVarianceClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision mass(\textcolor{red}{\textless double precision\textgreater} rootVariance\argin)
Return the mass corresponding to the given rootVariance of the cosmological density field.

type(varying_string) objectType() Return the type of the object.

double precision powerNormalization() Return the normalization of the power spectrum.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

double precision rootVariance(\textcolor{red}{\textless double precision\textgreater} mass\argin) Return the root-variance of the cosmological density field.

void rootVarianceAndLogarithmicGradient(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} rootVariance\argout,\textcolor{r double precision\textgreater} rootVarianceLogarithmicGradient\argout) Return the value and logarithmic gradient with respect to mass of the root-variance of the cosmological density field.

double precision rootVarianceLogarithmicGradient(\textcolor{red}{\textless double precision\textgreate mass\argin) Return the logarithmic gradient of the root-variance of the cosmological density field with respect to mass.

double precision sigma8() Return the value of σ_8.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

cosmologicalMassVarianceFilteredPower

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((cosmologicalMassVarianceClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision mass(\textcolor{red}{\textless double precision\textgreater} rootVariance\argin) Return the mass corresponding to the given rootVariance of the cosmological density field.

type(varying_string) objectType() Return the type of the object.

double precision powerNormalization() Return the normalization of the power spectrum.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void retabulate(<double> time→) Tabulate cosmological mass variance.

double precision rootVariance(\textcolor{red}{\textless double precision\textgreater} mass\argin) Return the root-variance of the cosmological density field.

void rootVarianceAndLogarithmicGradient(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} rootVariance\argout,\textcolor{r double precision\textgreater} rootVarianceLogarithmicGradient\argout) Return the value and logarithmic gradient with respect to mass of the root-variance of the cosmological density field.

double precision rootVarianceLogarithmicGradient(\textcolor{red}{\textless double precision\textgreate mass\argin) Return the logarithmic gradient of the root-variance of the cosmological density field with respect to mass.

double precision sigma8() Return the value of σ_8.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

cosmologicalMassVariancePeakBackgroundSplit

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((cosmologicalMassVarianceClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision mass(\textcolor{red}{\textless double precision\textgreater} rootVariance\argin) Return the mass corresponding to the given rootVariance of the cosmological density field.

type(varying_string) objectType() Return the type of the object.

double precision powerNormalization() Return the normalization of the power spectrum.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`double precision rootVariance(\textcolor{red}{\textless double precision\textgreater} mass\argin)` Return the root-variance of the cosmological density field.

`void rootVarianceAndLogarithmicGradient(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} rootVariance\argout,\textcolor{r double precision\textgreater} rootVarianceLogarithmicGradient\argout)` Return the value and logarithmic gradient with respect to mass of the root-variance of the cosmological density field.

`double precision rootVarianceLogarithmicGradient(\textcolor{red}{\textless double precision\textgreate mass\argin)` Return the logarithmic gradient of the root-variance of the cosmological density field with respect to mass.

`double precision sigma8()` Return the value of σ_8.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

cosmologyFunctionsClass

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`double precision comovingVolumeElementRedshift(\textcolor{red}{\textless double precision\textgreater} time\argin)` Returns the differential comoving volume element $\mathrm{d}V/\mathrm{d}z = r_\mathrm{c}^2(t)\mathrm{c}H^{-1}(t)$ (where r_c is the comoving distance to time t and $H(t)$ is the Hubble parameter at that time) for unit solid angle at the specified `time`.

`double precision comovingVolumeElementTime(\textcolor{red}{\textless double precision\textgreater} time\argin)` Returns the differential comoving volume element $\mathrm{d}V/\mathrm{d}t = r_\mathrm{c}^2(t)\mathrm{c}a(t)$ (where r_c is the comoving distance to time t and $a(t)$ is the expansion at that time) for unit solid angle at the specified `time`.

`double precision cosmicTime(\textcolor{red}{\textless double precision\textgreater} expansionFactor\arg logical\textgreater} collapsingPhase\argin)` Return the cosmological age at the given expansion factor.

`void deepCopy(\textcolor{red}{\textless class((cosmologyFunctionsClass))\textgreater} destination\argin` Perform a deep copy of the object.

`void densityScalingEarlyTime(\textcolor{red}{\textless double precision\textgreater} dominateFactor\arg double precision\textgreater} densityPower\argout,\textcolor{red}{\textless double precision\textgreater} expansionFactorDominant\argout,\textcolor{red}{\textless double precision\textgreater} OmegaDominant\argout)` Compute the scaling of density with expansion factor at early times in the universe.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision distanceAngular(\textcolor{red}{\textless double precision\textgreater} time\argin)` Return the angular diameter distance to the given cosmic `time`.

`double precision distanceComoving(\textcolor{red}{\textless double precision\textgreater} time\argin)` Return the comoving distance to the given cosmic `time`.

`double precision distanceComovingConvert(\textcolor{red}{\textless integer\textgreater} output\argin,\textcolor{red}{\textless double precision\textgreater} distanceLuminosity\argin,\text double precision\textgreater} distanceModulus\argin,\textcolor{red}{\textless double precision\textgreater} distanceModulusKCorrected\argin,\textcolor{red}{\textless double precision\textgreater} redshift\argin)` Convert between different measures of comoving distance.

`double precision distanceLuminosity(\textcolor{red}{\textless double precision\textgreater} time\argin)` Return the luminosity distance to the given cosmic `time`.

`double precision dominationEpochMatter(\textcolor{red}{\textless double precision\textgreater} dominateFactor\argin)` Compute the epoch at which matter dominates over other forms of energy by a given factor.

`double precision epochTime(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolo double precision\textgreater} expansionFactor\argin,\textcolor{red}{\textless logical\textgreater} collapsingPhase\argin)` Convenience function that returns the time corresponding to an epoch specified by time or expansion factor.

`void epochValidate(\textcolor{red}{\textless double precision\textgreater} timeIn \argin,\textcolor{red double precision\textgreater} expansionFactorIn \argin,\textcolor{red}{\textless logical\textgreate collapsingIn \argin,\textcolor{red}{\textless double precision\textgreater} timeOut \argout,\textcolor{red}{\textless double precision\textgreater} expansionFactorOut\argout,\textcolo logical\textgreater} collapsingOut \argout)` Check the given cosmic epoch is valid (aborting otherwise) and, optionally, return time or expansion factor associated with the epoch.

`double precision equalityEpochMatterCurvature(\textcolor{red}{\textless integer\textgreater} requestType\argin)` Return the epoch of matter-curvature magnitude equality (either expansion factor or cosmic time).

`double precision equalityEpochMatterDarkEnergy(\textcolor{red}{\textless integer\textgreater} requestType\argin)` Return the epoch of matter-dark energy magnitude equality (either expansion factor or cosmic time).

`double precision equalityEpochMatterRadiation(\textcolor{red}{\textless integer\textgreater} requestType\argin)` Return the epoch of matter-radiation magnitude equality (either expansion factor or cosmic time).

`double precision equationOfStateDarkEnergy(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin)` HASH(0x1fe6c28)

`double precision expansionFactor(\textcolor{red}{\textless double precision\textgreater} time\argin)` Returns the expansion factor at cosmological time `time`.

double precision expansionFactorFromRedshift(\textcolor{red}{\textless double precision\textgreater} redshift\argin) Returns expansion factor given a redshift.

double precision expansionRate(\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin) Returns the cosmological expansion rate, $\dot{a}/a$ at expansion factor expansionFactor.

double precision exponentDarkEnergy(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin) HASH(0x1fea6a0)

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

double precision hubbleParameterEpochal(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin,\textcolor logical\textgreater} collapsingPhase\argin) Returns the Hubble parameter at the requested cosmological time, time, or expansion factor, expansionFactor.

double precision hubbleParameterRateOfChange(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin,\textcolor logical\textgreater} collapsingPhase\argin) Returns the rate of change of the Hubble parameter at the requested cosmological time, time, or expansion factor, expansionFactor.

<logical> isDefault() Return true if this is the default object of this class.

double precision matterDensityEpochal(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin,\textcolor logical\textgreater} collapsingPhase\argin) Convenience function that returns the matter density at the specified epoch.

type(varying_string) objectType() Return the type of the object.

double precision omegaDarkEnergyEpochal(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin,\textcolor logical\textgreater} collapsingPhase\argin) Return the dark energy density parameter at expansion factor expansionFactor.

double precision omegaMatterEpochal(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin,\textcolor logical\textgreater} collapsingPhase\argin) Return the matter density parameter at expansion factor expansionFactor.

double precision omegaMatterRateOfChange(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin,\textcolor logical\textgreater} collapsingPhase\argin) Return the rate of change of the matter density parameter at expansion factor expansionFactor.

double precision redshiftFromExpansionFactor(\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin) Returns redshift for a given expansion factor.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision temperatureCMBEpochal(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin,\textcolor logical\textgreater} collapsingPhase\argin) Return the temperature of the cosmic microwave background at expansionFactor.

double precision timeAtDistanceComoving(\textcolor{red}{\textless double precision\textgreater} comovingDistance\argin) Return the cosmic time corresponding to the given comovingDistance.

double precision timeBigCrunch() Return the cosmological age at Big Crunch (or a negative value if no Big Crunch occurs).

cosmologyFunctionsMatterDarkEnergy

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

double precision comovingVolumeElementRedshift(\textcolor{red}{\textless double precision\textgreater} time\argin) Returns the differential comoving volume element $\mathrm{d}V/\mathrm{d}z = r_\mathrm{c}^2(t)\mathrm{c}H^{-1}(t)$ (where r_c is the comoving distance to time t and $H(t)$ is the Hubble parameter at that time) for unit solid angle at the specified time.

double precision comovingVolumeElementTime(\textcolor{red}{\textless double precision\textgreater} time\argin) Returns the differential comoving volume element $\mathrm{d}V/\mathrm{d}t = r_\mathrm{c}^2(t)\mathrm{c}a(t)$ (where r_c is the comoving distance to time t and $a(t)$ is the expansion at that time) for unit solid angle at the specified time.

double precision cosmicTime(\textcolor{red}{\textless double precision\textgreater} expansionFactor\arg logical\textgreater} collapsingPhase\argin) Return the cosmological age at the given expansion factor.

void deepCopy(\textcolor{red}{\textless class((cosmologyFunctionsClass))\textgreater} destination\argin Perform a deep copy of the object.

void densityScalingEarlyTime(\textcolor{red}{\textless double precision\textgreater} dominateFactor\arg double precision\textgreater} densityPower\argout,\textcolor{red}{\textless double precision\textgreater} expansionFactorDominant\argout,\textcolor{red}{\textless double precision\textgreater} OmegaDominant\argout) Compute the scaling of density with expansion factor at early times in the universe.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

`double precision distanceAngular(\textcolor{red}{\textless double precision\textgreater} time\argin)` Return the angular diameter distance to the given cosmic `time`.

`double precision distanceComoving(\textcolor{red}{\textless double precision\textgreater} time\argin)` Return the comoving distance to the given cosmic `time`.

`double precision distanceComovingConvert(\textcolor{red}{\textless integer\textgreater} output\argin,\textcolor{red}{\textless double precision\textgreater} distanceLuminosity\argin,\text double precision\textgreater} distanceModulus\argin,\textcolor{red}{\textless double precision\textgreater} distanceModulusKCorrected\argin,\textcolor{red}{\textless double precision\textgreater} redshift\argin)` Convert between different measures of comoving distance.

`double precision distanceLuminosity(\textcolor{red}{\textless double precision\textgreater} time\argin)` Return the luminosity distance to the given cosmic `time`.

`void distanceTabulate(<double> time→)` Tabulate comoving distance as a function of cosmic time.

`double precision dominationEpochMatter(\textcolor{red}{\textless double precision\textgreater} dominateFactor\argin)` Compute the epoch at which matter dominates over other forms of energy by a given factor.

`double precision epochTime(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolo double precision\textgreater} expansionFactor\argin,\textcolor{red}{\textless logical\textgreater} collapsingPhase\argin)` Convenience function that returns the time corresponding to an epoch specified by time or expansion factor.

`void epochValidate(\textcolor{red}{\textless double precision\textgreater} timeIn \argin,\textcolor{red double precision\textgreater} expansionFactorIn \argin,\textcolor{red}{\textless logical\textgreate collapsingIn \argin,\textcolor{red}{\textless double precision\textgreater} timeOut \argout,\textcolor{red}{\textless double precision\textgreater} expansionFactorOut\argout,\textcolo logical\textgreater} collapsingOut \argout)` Check the given cosmic epoch is valid (aborting otherwise) and, optionally, return time or expansion factor associated with the epoch.

`double precision equalityEpochMatterCurvature(\textcolor{red}{\textless integer\textgreater} requestType\argin)` Return the epoch of matter-curvature magnitude equality (either expansion factor or cosmic time).

`double precision equalityEpochMatterDarkEnergy(\textcolor{red}{\textless integer\textgreater} requestType\argin)` Return the epoch of matter-dark energy magnitude equality (either expansion factor or cosmic time).

`double precision equalityEpochMatterRadiation(\textcolor{red}{\textless integer\textgreater} requestType\argin)` Return the epoch of matter-radiation magnitude equality (either expansion factor or cosmic time).

`double precision equationOfStateDarkEnergy(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin)` HASH(0x1fe6c28)

`double precision expansionFactor(\textcolor{red}{\textless double precision\textgreater} time\argin)` Returns the expansion factor at cosmological time `time`.

`double precision expansionFactorFromRedshift(\textcolor{red}{\textless double precision\textgreater} redshift\argin)` Returns expansion factor given a redshift.

`void expansionFactorTabulate(<double> time→)` Tabulate expansion factor as a function of cosmic time.

`double precision expansionRate(\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin)` Returns the cosmological expansion rate, $\dot{a}/a$ at expansion factor `expansionFactor`.

`double precision exponentDarkEnergy(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin)` HASH(0x1fea6a0)

`<double> exponentDarkEnergyDerivative(<double> [time]→, <double> [expansionFactor]→)` Return the derivative of the dark energy exponent with respect to expansion factor.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`double precision hubbleParameterEpochal(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin,\textcolor logical\textgreater} collapsingPhase\argin)` Returns the Hubble parameter at the requested cosmological time, `time`, or expansion factor, `expansionFactor`.

`double precision hubbleParameterRateOfChange(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin,\textcolor logical\textgreater} collapsingPhase\argin)` Returns the rate of change of the Hubble parameter at the requested cosmological time, `time`, or expansion factor, `expansionFactor`.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision matterDensityEpochal(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin,\textcolor logical\textgreater} collapsingPhase\argin)` Convenience function that returns the matter density at the specified epoch.

`type(varying_string) objectType()` Return the type of the object.

`double precision omegaDarkEnergyEpochal(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin,\textcolor logical\textgreater} collapsingPhase\argin)` Return the dark energy density parameter at expansion factor `expansionFactor`.

`double precision omegaMatterEpochal(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin,\textcolor logical\textgreater} collapsingPhase\argin)` Return the matter density parameter at expansion factor `expansionFactor`.

`double precision omegaMatterRateOfChange(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin,\textcolor logical\textgreater} collapsingPhase\argin)` Return the rate of change of the matter density parameter at expansion factor `expansionFactor`.

`double precision redshiftFromExpansionFactor(\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin)` Returns redshift for a given expansion factor.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

void targetSelf() Set a module-scope pointer to self.

double precision temperatureCMBEpochal(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin,\textcolor logical\textgreater} collapsingPhase\argin) Return the temperature of the cosmic microwave background at expansionFactor.

double precision timeAtDistanceComoving(\textcolor{red}{\textless double precision\textgreater} comovingDistance\argin) Return the cosmic time corresponding to the given comovingDistance.

double precision timeBigCrunch() Return the cosmological age at Big Crunch (or a negative value if no Big Crunch occurs).

cosmologyFunctionsMatterLambda

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

double precision comovingVolumeElementRedshift(\textcolor{red}{\textless double precision\textgreater} time\argin) Returns the differential comoving volume element $\mathrm{d}V/\mathrm{d}z = r_\mathrm{c}^2(t)\mathrm{c}H^{-1}(t)$ (where r_c is the comoving distance to time t and $H(t)$ is the Hubble parameter at that time) for unit solid angle at the specified time.

double precision comovingVolumeElementTime(\textcolor{red}{\textless double precision\textgreater} time\argin) Returns the differential comoving volume element $\mathrm{d}V/\mathrm{d}t = r_\mathrm{c}^2(t)\mathrm{c}a(t)$ (where r_c is the comoving distance to time t and $a(t)$ is the expansion at that time) for unit solid angle at the specified time.

double precision cosmicTime(\textcolor{red}{\textless double precision\textgreater} expansionFactor\arg logical\textgreater} collapsingPhase\argin) Return the cosmological age at the given expansion factor.

void deepCopy(\textcolor{red}{\textless class((cosmologyFunctionsClass))\textgreater} destination\argin Perform a deep copy of the object.

void densityScalingEarlyTime(\textcolor{red}{\textless double precision\textgreater} dominateFactor\arg double precision\textgreater} densityPower\argout,\textcolor{red}{\textless double precision\textgreater} expansionFactorDominant\argout,\textcolor{red}{\textless double precision\textgreater} OmegaDominant\argout) Compute the scaling of density with expansion factor at early times in the universe.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision distanceAngular(\textcolor{red}{\textless double precision\textgreater} time\argin)` Return the angular diameter distance to the given cosmic `time`.

`double precision distanceComoving(\textcolor{red}{\textless double precision\textgreater} time\argin)` Return the comoving distance to the given cosmic `time`.

`double precision distanceComovingConvert(\textcolor{red}{\textless integer\textgreater} output\argin,\textcolor{red}{\textless double precision\textgreater} distanceLuminosity\argin,\text double precision\textgreater} distanceModulus\argin,\textcolor{red}{\textless double precision\textgreater} distanceModulusKCorrected\argin,\textcolor{red}{\textless double precision\textgreater} redshift\argin)` Convert between different measures of comoving distance.

`double precision distanceLuminosity(\textcolor{red}{\textless double precision\textgreater} time\argin)` Return the luminosity distance to the given cosmic `time`.

`void distanceTabulate(<double> time→)` Tabulate comoving distance as a function of cosmic time.

`double precision dominationEpochMatter(\textcolor{red}{\textless double precision\textgreater} dominateFactor\argin)` Compute the epoch at which matter dominates over other forms of energy by a given factor.

`double precision epochTime(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolo double precision\textgreater} expansionFactor\argin,\textcolor{red}{\textless logical\textgreater} collapsingPhase\argin)` Convenience function that returns the time corresponding to an epoch specified by time or expansion factor.

`void epochValidate(\textcolor{red}{\textless double precision\textgreater} timeIn \argin,\textcolor{red double precision\textgreater} expansionFactorIn \argin,\textcolor{red}{\textless logical\textgreate collapsingIn \argin,\textcolor{red}{\textless double precision\textgreater} timeOut \argout,\textcolor{red}{\textless double precision\textgreater} expansionFactorOut\argout,\textcolo logical\textgreater} collapsingOut \argout)` Check the given cosmic epoch is valid (aborting otherwise) and, optionally, return time or expansion factor associated with the epoch.

`double precision equalityEpochMatterCurvature(\textcolor{red}{\textless integer\textgreater} requestType\argin)` Return the epoch of matter-curvature magnitude equality (either expansion factor or cosmic time).

`double precision equalityEpochMatterDarkEnergy(\textcolor{red}{\textless integer\textgreater} requestType\argin)` Return the epoch of matter-dark energy magnitude equality (either expansion factor or cosmic time).

`double precision equalityEpochMatterRadiation(\textcolor{red}{\textless integer\textgreater} requestType\argin)` Return the epoch of matter-radiation magnitude equality (either expansion factor or cosmic time).

`double precision equationOfStateDarkEnergy(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin)` HASH(0x1fe6c28)

`double precision expansionFactor(\textcolor{red}{\textless double precision\textgreater} time\argin)` Returns the expansion factor at cosmological time `time`.

`double precision expansionFactorFromRedshift(\textcolor{red}{\textless double precision\textgreater} redshift\argin)` Returns expansion factor given a redshift.

`void expansionFactorTabulate(<double> time→)` Tabulate expansion factor as a function of cosmic time.

`double precision expansionRate(\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin)` Returns the cosmological expansion rate, $\dot{a}/a$ at expansion factor `expansionFactor`.

`double precision exponentDarkEnergy(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin)` HASH(0x1fea6a0)

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`double precision hubbleParameterEpochal(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin,\textcolor logical\textgreater} collapsingPhase\argin)` Returns the Hubble parameter at the requested cosmological time, `time`, or expansion factor, `expansionFactor`.

`double precision hubbleParameterRateOfChange(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin,\textcolor logical\textgreater} collapsingPhase\argin)` Returns the rate of change of the Hubble parameter at the requested cosmological time, `time`, or expansion factor, `expansionFactor`.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision matterDensityEpochal(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin,\textcolor logical\textgreater} collapsingPhase\argin)` Convenience function that returns the matter density at the specified epoch.

`type(varying_string) objectType()` Return the type of the object.

`double precision omegaDarkEnergyEpochal(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin,\textcolor logical\textgreater} collapsingPhase\argin)` Return the dark energy density parameter at expansion factor `expansionFactor`.

`double precision omegaMatterEpochal(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin,\textcolor logical\textgreater} collapsingPhase\argin)` Return the matter density parameter at expansion factor `expansionFactor`.

`double precision omegaMatterRateOfChange(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin,\textcolor logical\textgreater} collapsingPhase\argin)` Return the rate of change of the matter density parameter at expansion factor `expansionFactor`.

`double precision redshiftFromExpansionFactor(\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin)` Returns redshift for a given expansion factor.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision temperatureCMBEpochal(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin,\textcolor logical\textgreater} collapsingPhase\argin) Return the temperature of the cosmic microwave background at expansionFactor.

double precision timeAtDistanceComoving(\textcolor{red}{\textless double precision\textgreater} comovingDistance\argin) Return the cosmic time corresponding to the given comovingDistance.

double precision timeBigCrunch() Return the cosmological age at Big Crunch (or a negative value if no Big Crunch occurs).

cosmologyFunctionsStaticUniverse

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

double precision comovingVolumeElementRedshift(\textcolor{red}{\textless double precision\textgreater} time\argin) Returns the differential comoving volume element $\mathrm{d}V/\mathrm{d}z = r_\mathrm{c}^2(t)\mathrm{c}H^{-1}(t)$ (where r_c is the comoving distance to time t and $H(t)$ is the Hubble parameter at that time) for unit solid angle at the specified time.

double precision comovingVolumeElementTime(\textcolor{red}{\textless double precision\textgreater} time\argin) Returns the differential comoving volume element $\mathrm{d}V/\mathrm{d}t = r_\mathrm{c}^2(t)\mathrm{c}a(t)$ (where r_c is the comoving distance to time t and $a(t)$ is the expansion at that time) for unit solid angle at the specified time.

double precision cosmicTime(\textcolor{red}{\textless double precision\textgreater} expansionFactor\arg logical\textgreater} collapsingPhase\argin) Return the cosmological age at the given expansion factor.

void deepCopy(\textcolor{red}{\textless class((cosmologyFunctionsClass))\textgreater} destination\argin Perform a deep copy of the object.

void densityScalingEarlyTime(\textcolor{red}{\textless double precision\textgreater} dominateFactor\arg double precision\textgreater} densityPower\argout,\textcolor{red}{\textless double precision\textgreater} expansionFactorDominant\argout,\textcolor{red}{\textless double precision\textgreater} OmegaDominant\argout) Compute the scaling of density with expansion factor at early times in the universe.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision distanceAngular(\textcolor{red}{\textless double precision\textgreater} time\argin)` Return the angular diameter distance to the given cosmic `time`.

`double precision distanceComoving(\textcolor{red}{\textless double precision\textgreater} time\argin)` Return the comoving distance to the given cosmic `time`.

`double precision distanceComovingConvert(\textcolor{red}{\textless integer\textgreater} output\argin,\textcolor{red}{\textless double precision\textgreater} distanceLuminosity\argin,\text double precision\textgreater} distanceModulus\argin,\textcolor{red}{\textless double precision\textgreater} distanceModulusKCorrected\argin,\textcolor{red}{\textless double precision\textgreater} redshift\argin)` Convert between different measures of comoving distance.

`double precision distanceLuminosity(\textcolor{red}{\textless double precision\textgreater} time\argin)` Return the luminosity distance to the given cosmic `time`.

`double precision dominationEpochMatter(\textcolor{red}{\textless double precision\textgreater} dominateFactor\argin)` Compute the epoch at which matter dominates over other forms of energy by a given factor.

`double precision epochTime(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolo double precision\textgreater} expansionFactor\argin,\textcolor{red}{\textless logical\textgreater} collapsingPhase\argin)` Convenience function that returns the time corresponding to an epoch specified by time or expansion factor.

`void epochValidate(\textcolor{red}{\textless double precision\textgreater} timeIn \argin,\textcolor{red double precision\textgreater} expansionFactorIn \argin,\textcolor{red}{\textless logical\textgreate collapsingIn \argin,\textcolor{red}{\textless double precision\textgreater} timeOut \argout,\textcolor{red}{\textless double precision\textgreater} expansionFactorOut\argout,\textcolo logical\textgreater} collapsingOut \argout)` Check the given cosmic epoch is valid (aborting otherwise) and, optionally, return time or expansion factor associated with the epoch.

`double precision equalityEpochMatterCurvature(\textcolor{red}{\textless integer\textgreater} requestType\argin)` Return the epoch of matter-curvature magnitude equality (either expansion factor or cosmic time).

`double precision equalityEpochMatterDarkEnergy(\textcolor{red}{\textless integer\textgreater} requestType\argin)` Return the epoch of matter-dark energy magnitude equality (either expansion factor or cosmic time).

`double precision equalityEpochMatterRadiation(\textcolor{red}{\textless integer\textgreater} requestType\argin)` Return the epoch of matter-radiation magnitude equality (either expansion factor or cosmic time).

`double precision equationOfStateDarkEnergy(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin)` HASH(0x1fe6c28)

`double precision expansionFactor(\textcolor{red}{\textless double precision\textgreater} time\argin)` Returns the expansion factor at cosmological time `time`.

double precision expansionFactorFromRedshift(\textcolor{red}{\textless double precision\textgreater} redshift\argin) Returns expansion factor given a redshift.

double precision expansionRate(\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin) Returns the cosmological expansion rate, $\dot{a}/a$ at expansion factor expansionFactor.

double precision exponentDarkEnergy(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin) HASH(0x1fea6a0)

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

double precision hubbleParameterEpochal(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin,\textcolor logical\textgreater} collapsingPhase\argin) Returns the Hubble parameter at the requested cosmological time, time, or expansion factor, expansionFactor.

double precision hubbleParameterRateOfChange(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin,\textcolor logical\textgreater} collapsingPhase\argin) Returns the rate of change of the Hubble parameter at the requested cosmological time, time, or expansion factor, expansionFactor.

<logical> isDefault() Return true if this is the default object of this class.

double precision matterDensityEpochal(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin,\textcolor logical\textgreater} collapsingPhase\argin) Convenience function that returns the matter density at the specified epoch.

type(varying_string) objectType() Return the type of the object.

double precision omegaDarkEnergyEpochal(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin,\textcolor logical\textgreater} collapsingPhase\argin) Return the dark energy density parameter at expansion factor expansionFactor.

double precision omegaMatterEpochal(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin,\textcolor logical\textgreater} collapsingPhase\argin) Return the matter density parameter at expansion factor expansionFactor.

double precision omegaMatterRateOfChange(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin,\textcolor logical\textgreater} collapsingPhase\argin) Return the rate of change of the matter density parameter at expansion factor expansionFactor.

double precision redshiftFromExpansionFactor(\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin) Returns redshift for a given expansion factor.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision temperatureCMBEpochal(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin,\textcolor logical\textgreater} collapsingPhase\argin) Return the temperature of the cosmic microwave background at expansionFactor.

double precision timeAtDistanceComoving(\textcolor{red}{\textless double precision\textgreater} comovingDistance\argin) Return the cosmic time corresponding to the given comovingDistance.

double precision timeBigCrunch() Return the cosmological age at Big Crunch (or a negative value if no Big Crunch occurs).

cosmologyParametersClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((cosmologyParametersClass))\textgreater} destination\arginout) Perform a deep copy of the object.

double precision densityCritical() Return the critical density at the present day in units of $M_\odot/\mathrm{Mpc}^3$.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

double precision HubbleConstant(\textcolor{red}{\textless integer\textgreater} units\argin) Return the Hubble constant at the present day. The optional units argument specifies if the return value should be in units of km/s/Mpc (hubbleUnitsStandard), Gyr^{-1} (hubbleUnitsTime), or 100 km/s/Mpc (hubbleUnitsLittleH).

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision OmegaBaryon() Return the cosmological baryon density in units of the critical density at the present day.

double precision OmegaCurvature() Return the cosmological curvature density in units of the critical density at the present day.

`double precision OmegaDarkEnergy()` Return the cosmological dark energy density in units of the critical density at the present day.

`double precision OmegaMatter()` Return the cosmological matter density in units of the critical density at the present day.

`double precision OmegaRadiation()` Return the cosmological radiation density in units of the critical density at the present day.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision temperatureCMB()` Return the temperature of the cosmic microwave background radiation (in units of Kelvin) at the present day.

cosmologyParametersSimple

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((cosmologyParametersClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`double precision densityCritical()` Return the critical density at the present day in units of $M_\odot/\mathrm{Mpc}^3$.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`double precision HubbleConstant(\textcolor{red}{\textless integer\textgreater} units\argin)` Return the Hubble constant at the present day. The optional `units` argument specifies if the return value should be in units of km/s/Mpc (hubbleUnitsStandard), Gyr^{-1} (hubbleUnitsTime), or 100 km/s/Mpc (hubbleUnitsLittleH).

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`double precision OmegaBaryon()` Return the cosmological baryon density in units of the critical density at the present day.

`double precision OmegaCurvature()` Return the cosmological curvature density in units of the critical density at the present day.

`double precision OmegaDarkEnergy()` Return the cosmological dark energy density in units of the critical density at the present day.

`double precision OmegaMatter()` Return the cosmological matter density in units of the critical density at the present day.

`double precision OmegaRadiation()` Return the cosmological radiation density in units of the critical density at the present day.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision temperatureCMB()` Return the temperature of the cosmic microwave background radiation (in units of Kelvin) at the present day.

criticalOverdensityBarkana2001WDM

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`double precision collapsingMass(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin,\textcolor logical\textgreater} collapsing \argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout)` Return the mass scale just collapsing at the given cosmic time.

`void deepCopy(\textcolor{red}{\textless class((criticalOverdensityClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

double precision gradientMass(\textcolor{red}{\textless double precision\textgreater} time\argin,\textc
double precision\textgreater} expansionFactor\argin,\textcolor{red}{\textless logical\textgreater}
collapsing \argin,\textcolor{red}{\textless double precision\textgreater} mass \argin,\textcolor{re
type((treeNode))\textgreater} node \arginout) Return the derivative with respect to mass of the linear theory critical overdensity for collapse at the given cosmic time.

double precision gradientTime(\textcolor{red}{\textless double precision\textgreater} time\argin,\textc
double precision\textgreater} expansionFactor\argin,\textcolor{red}{\textless logical\textgreater}
collapsing \argin,\textcolor{red}{\textless double precision\textgreater} mass \argin,\textcolor{re
type((treeNode))\textgreater} node \arginout) Return the derivative with respect to time of the linear theory critical overdensity for collapse at the given cosmic time.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

logical isMassDependent() Return true if the critical overdensity is dependent on the mass of the halo.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision timeOfCollapse(\textcolor{red}{\textless double precision\textgreater}
criticalOverdensity\argin,\textcolor{red}{\textless double precision\textgreater} mass
\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout) Returns the time of collapse for a perturbation of linear theory overdensity criticalOverdensity.

double precision value(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{re
double precision\textgreater} expansionFactor\argin,\textcolor{red}{\textless logical\textgreater}
collapsing \argin,\textcolor{red}{\textless double precision\textgreater} mass \argin,\textcolor{re
type((treeNode))\textgreater} node \arginout) Return the critical overdensity at the given time and mass.

criticalOverdensityClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

double precision collapsingMass(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin,\textcolor{red}{\textless logical\textgreater} collapsing \argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout) Return the mass scale just collapsing at the given cosmic time.

void deepCopy(\textcolor{red}{\textless class((criticalOverdensityClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\textcolor{red}{\textless logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision gradientMass(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin,\textcolor{red}{\textless logical\textgreater} collapsing \argin,\textcolor{red}{\textless double precision\textgreater} mass \argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout) Return the derivative with respect to mass of the linear theory critical overdensity for collapse at the given cosmic time.

double precision gradientTime(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin,\textcolor{red}{\textless logical\textgreater} collapsing \argin,\textcolor{red}{\textless double precision\textgreater} mass \argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout) Return the derivative with respect to time of the linear theory critical overdensity for collapse at the given cosmic time.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDigest) Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

logical isMassDependent() Return true if the critical overdensity is dependent on the mass of the halo.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision timeOfCollapse(\textcolor{red}{\textless double precision\textgreater} criticalOverdensity\argin,\textcolor{red}{\textless double precision\textgreater} mass \argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout) Returns the time of collapse for a perturbation of linear theory overdensity criticalOverdensity.

double precision value(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{re
double precision\textgreater} expansionFactor\argin,\textcolor{red}{\textless logical\textgreater}
collapsing \argin,\textcolor{red}{\textless double precision\textgreater} mass \argin,\textcolor{re
type((treeNode))\textgreater} node \arginout) Return the critical overdensity at the given time and mass.

criticalOverdensityEnvironmental

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

double precision collapsingMass(\textcolor{red}{\textless double precision\textgreater}
time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin,\textcolor
logical\textgreater} collapsing \argin,\textcolor{red}{\textless type((treeNode))\textgreater}
node \arginout) Return the mass scale just collapsing at the given cosmic time.

void deepCopy(\textcolor{red}{\textless class((criticalOverdensityClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision gradientMass(\textcolor{red}{\textless double precision\textgreater} time\argin,\textc
double precision\textgreater} expansionFactor\argin,\textcolor{red}{\textless logical\textgreater}
collapsing \argin,\textcolor{red}{\textless double precision\textgreater} mass \argin,\textcolor{re
type((treeNode))\textgreater} node \arginout) Return the derivative with respect to mass of the linear theory critical overdensity for collapse at the given cosmic time.

double precision gradientTime(\textcolor{red}{\textless double precision\textgreater} time\argin,\textc
double precision\textgreater} expansionFactor\argin,\textcolor{red}{\textless logical\textgreater}
collapsing \argin,\textcolor{red}{\textless double precision\textgreater} mass \argin,\textcolor{re
type((treeNode))\textgreater} node \arginout) Return the derivative with respect to time of the linear theory critical overdensity for collapse at the given cosmic time.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

logical isMassDependent() Return true if the critical overdensity is dependent on the mass of the halo.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision timeOfCollapse(\textcolor{red}{\textless double precision\textgreater} criticalOverdensity\argin,\textcolor{red}{\textless double precision\textgreater} mass \argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout) Returns the time of collapse for a perturbation of linear theory overdensity criticalOverdensity.

double precision value(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{re double precision\textgreater} expansionFactor\argin,\textcolor{red}{\textless logical\textgreater} collapsing \argin,\textcolor{red}{\textless double precision\textgreater} mass \argin,\textcolor{re type((treeNode))\textgreater} node \arginout) Return the critical overdensity at the given time and mass.

criticalOverdensityFixed

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

double precision collapsingMass(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin,\textcolor logical\textgreater} collapsing \argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout) Return the mass scale just collapsing at the given cosmic time.

void deepCopy(\textcolor{red}{\textless class((criticalOverdensityClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision gradientMass(\textcolor{red}{\textless double precision\textgreater} time\argin,\texto double precision\textgreater} expansionFactor\argin,\textcolor{red}{\textless logical\textgreater} collapsing \argin,\textcolor{red}{\textless double precision\textgreater} mass \argin,\textcolor{re type((treeNode))\textgreater} node \arginout) Return the derivative with respect to mass of the linear theory critical overdensity for collapse at the given cosmic time.

double precision gradientTime(\textcolor{red}{\textless double precision\textgreater} time\argin,\texto double precision\textgreater} expansionFactor\argin,\textcolor{red}{\textless logical\textgreater} collapsing \argin,\textcolor{red}{\textless double precision\textgreater} mass \argin,\textcolor{re type((treeNode))\textgreater} node \arginout) Return the derivative with respect to time of the linear theory critical overdensity for collapse at the given cosmic time.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

logical isMassDependent() Return true if the critical overdensity is dependent on the mass of the halo.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision timeOfCollapse(\textcolor{red}{\textless double precision\textgreater} criticalOverdensity\argin,\textcolor{red}{\textless double precision\textgreater} mass \argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout) Returns the time of collapse for a perturbation of linear theory overdensity criticalOverdensity.

double precision value(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{re double precision\textgreater} expansionFactor\argin,\textcolor{red}{\textless logical\textgreater} collapsing \argin,\textcolor{red}{\textless double precision\textgreater} mass \argin,\textcolor{re type((treeNode))\textgreater} node \arginout) Return the critical overdensity at the given time and mass.

criticalOverdensityKitayamaSuto1996

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

double precision collapsingMass(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin,\textcolor logical\textgreater} collapsing \argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout) Return the mass scale just collapsing at the given cosmic time.

void deepCopy(\textcolor{red}{\textless class((criticalOverdensityClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

`double precision gradientMass(\textcolor{red}{\textless double precision\textgreater} time\argin,\texto double precision\textgreater} expansionFactor\argin,\textcolor{red}{\textless logical\textgreater} collapsing \argin,\textcolor{red}{\textless double precision\textgreater} mass \argin,\textcolor{re type((treeNode))\textgreater} node \arginout)` Return the derivative with respect to mass of the linear theory critical overdensity for collapse at the given cosmic time.

`double precision gradientTime(\textcolor{red}{\textless double precision\textgreater} time\argin,\texto double precision\textgreater} expansionFactor\argin,\textcolor{red}{\textless logical\textgreater} collapsing \argin,\textcolor{red}{\textless double precision\textgreater} mass \argin,\textcolor{re type((treeNode))\textgreater} node \arginout)` Return the derivative with respect to time of the linear theory critical overdensity for collapse at the given cosmic time.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`logical isMassDependent()` Return true if the critical overdensity is dependent on the mass of the halo.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision timeOfCollapse(\textcolor{red}{\textless double precision\textgreater} criticalOverdensity\argin,\textcolor{red}{\textless double precision\textgreater} mass \argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout)` Returns the time of collapse for a perturbation of linear theory overdensity `criticalOverdensity`.

`double precision value(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{re double precision\textgreater} expansionFactor\argin,\textcolor{red}{\textless logical\textgreater} collapsing \argin,\textcolor{red}{\textless double precision\textgreater} mass \argin,\textcolor{re type((treeNode))\textgreater} node \arginout)` Return the critical overdensity at the given time and mass.

criticalOverdensityPeakBackgroundSplit

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

double precision collapsingMass(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin,\textcolor{red}{\textless logical\textgreater} collapsing \argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout) Return the mass scale just collapsing at the given cosmic time.

void deepCopy(\textcolor{red}{\textless class((criticalOverdensityClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\textcolor{red}{\textless logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision gradientMass(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin,\textcolor{red}{\textless logical\textgreater} collapsing \argin,\textcolor{red}{\textless double precision\textgreater} mass \argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout) Return the derivative with respect to mass of the linear theory critical overdensity for collapse at the given cosmic time.

double precision gradientTime(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin,\textcolor{red}{\textless logical\textgreater} collapsing \argin,\textcolor{red}{\textless double precision\textgreater} mass \argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout) Return the derivative with respect to time of the linear theory critical overdensity for collapse at the given cosmic time.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDigest) Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

logical isMassDependent() Return true if the critical overdensity is dependent on the mass of the halo.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision timeOfCollapse(\textcolor{red}{\textless double precision\textgreater} criticalOverdensity\argin,\textcolor{red}{\textless double precision\textgreater} mass \argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout) Returns the time of collapse for a perturbation of linear theory overdensity criticalOverdensity.

double precision value(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{re
double precision\textgreater} expansionFactor\argin,\textcolor{red}{\textless logical\textgreater}
collapsing \argin,\textcolor{red}{\textless double precision\textgreater} mass \argin,\textcolor{re
type((treeNode))\textgreater} node \arginout) Return the critical overdensity at the given time and mass.

criticalOverdensitySphericalCollapseMatterDE

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

double precision collapsingMass(\textcolor{red}{\textless double precision\textgreater}
time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin,\textcolor
logical\textgreater} collapsing \argin,\textcolor{red}{\textless type((treeNode))\textgreater}
node \arginout) Return the mass scale just collapsing at the given cosmic time.

void deepCopy(\textcolor{red}{\textless class((criticalOverdensityClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision gradientMass(\textcolor{red}{\textless double precision\textgreater} time\argin,\textc
double precision\textgreater} expansionFactor\argin,\textcolor{red}{\textless logical\textgreater}
collapsing \argin,\textcolor{red}{\textless double precision\textgreater} mass \argin,\textcolor{re
type((treeNode))\textgreater} node \arginout) Return the derivative with respect to mass of the linear theory critical overdensity for collapse at the given cosmic time.

double precision gradientTime(\textcolor{red}{\textless double precision\textgreater} time\argin,\textc
double precision\textgreater} expansionFactor\argin,\textcolor{red}{\textless logical\textgreater}
collapsing \argin,\textcolor{red}{\textless double precision\textgreater} mass \argin,\textcolor{re
type((treeNode))\textgreater} node \arginout) Return the derivative with respect to time of the linear theory critical overdensity for collapse at the given cosmic time.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

logical isMassDependent() Return true if the critical overdensity is dependent on the mass of the halo.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

`void retabulate(<double> time→)` Tabulate spherical collapse critical overdensity.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision timeOfCollapse(\textcolor{red}{\textless double precision\textgreater} criticalOverdensity\argin,\textcolor{red}{\textless double precision\textgreater} mass \argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout)` Returns the time of collapse for a perturbation of linear theory overdensity `criticalOverdensity`.

`double precision value(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{re double precision\textgreater} expansionFactor\argin,\textcolor{red}{\textless logical\textgreater} collapsing \argin,\textcolor{red}{\textless double precision\textgreater} mass \argin,\textcolor{re type((treeNode))\textgreater} node \arginout)` Return the critical overdensity at the given time and mass.

criticalOverdensitySphericalCollapseMatterLambda

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`double precision collapsingMass(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin,\textcolor logical\textgreater} collapsing \argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout)` Return the mass scale just collapsing at the given cosmic time.

`void deepCopy(\textcolor{red}{\textless class((criticalOverdensityClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision gradientMass(\textcolor{red}{\textless double precision\textgreater} time\argin,\texto double precision\textgreater} expansionFactor\argin,\textcolor{red}{\textless logical\textgreater} collapsing \argin,\textcolor{red}{\textless double precision\textgreater} mass \argin,\textcolor{re type((treeNode))\textgreater} node \arginout)` Return the derivative with respect to mass of the linear theory critical overdensity for collapse at the given cosmic time.

`double precision gradientTime(\textcolor{red}{\textless double precision\textgreater} time\argin,\texto double precision\textgreater} expansionFactor\argin,\textcolor{red}{\textless logical\textgreater} collapsing \argin,\textcolor{red}{\textless double precision\textgreater} mass \argin,\textcolor{re type((treeNode))\textgreater} node \arginout)` Return the derivative with respect to time of the linear theory critical overdensity for collapse at the given cosmic time.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

logical isMassDependent() Return true if the critical overdensity is dependent on the mass of the halo.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void retabulate(<double> time→) Tabulate spherical collapse critical overdensity.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision timeOfCollapse(\textcolor{red}{\textless double precision\textgreater}
criticalOverdensity\argin,\textcolor{red}{\textless double precision\textgreater} mass
\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout) Returns the time of collapse for a perturbation of linear theory overdensity criticalOverdensity.

double precision value(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{re
double precision\textgreater} expansionFactor\argin,\textcolor{red}{\textless logical\textgreater}
collapsing \argin,\textcolor{red}{\textless double precision\textgreater} mass \argin,\textcolor{re
type((treeNode))\textgreater} node \arginout) Return the critical overdensity at the given time and mass.

darkMatterHaloBiasClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

double precision bias(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red
double precision\textgreater} time\argin | \textcolor{red}{\textless type((treeNode))\textgreater}
node\arginout) Returns the bias of a halo specified by a mass (in $M_\odot$) and time (in Gyr). | Returns the bias of the halo in the supplied node.

void deepCopy(\textcolor{red}{\textless class((darkMatterHaloBiasClass))\textgreater} destination\argin
Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

darkMatterHaloBiasPressSchechter

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`double precision bias(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red double precision\textgreater} time\argin | \textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Returns the bias of a halo specified by a mass (in $M_\odot$) and time (in Gyr). | Returns the bias of the halo in the supplied node.

`void deepCopy(\textcolor{red}{\textless class((darkMatterHaloBiasClass))\textgreater} destination\argin` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

darkMatterHaloBiasSheth2001

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

double precision bias(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red} double precision\textgreater} time\argin | \textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the bias of a halo specified by a mass (in $M_\odot$) and time (in Gyr). | Returns the bias of the halo in the supplied node.

void deepCopy(\textcolor{red}{\textless class((darkMatterHaloBiasClass))\textgreater} destination\argin Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

darkMatterHaloBiasTinker2010

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

double precision bias(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red} double precision\textgreater} time\argin | \textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the bias of a halo specified by a mass (in $M_\odot$) and time (in Gyr). | Returns the bias of the halo in the supplied node.

void deepCopy(\textcolor{red}{\textless class((darkMatterHaloBiasClass))\textgreater} destination\argin Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

darkMatterHaloMassAccretionHistoryClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((darkMatterHaloMassAccretionHistoryClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision time(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{
double precision\textgreater} mass\argin) Returns the time at which the given halo mass was reached.

darkMatterHaloMassAccretionHistoryCorrea2015

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((darkMatterHaloMassAccretionHistoryClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision time(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{ double precision\textgreater} mass\argin) Returns the time at which the given halo mass was reached.

darkMatterHaloMassAccretionHistoryWechsler2002

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((darkMatterHaloMassAccretionHistoryClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision time(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{ double precision\textgreater} mass\argin) Returns the time at which the given halo mass was reached.

darkMatterHaloMassAccretionHistoryZhao2009

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((darkMatterHaloMassAccretionHistoryClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision time(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{
double precision\textgreater} mass\argin) Returns the time at which the given halo mass was reached.

darkMatterHaloMassLossRateClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((darkMatterHaloMassLossRateClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision rate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)
Returns the rate of mass loss (in $M_\odot$/Gyr) from node.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

darkMatterHaloMassLossRateVanDenBosch

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((darkMatterHaloMassLossRateClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision rate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)
Returns the rate of mass loss (in $M_\odot$/Gyr) from node.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

darkMatterHaloMassLossRateZero

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((darkMatterHaloMassLossRateClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision rate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the rate of mass loss (in $M_\odot$/Gyr) from node.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

darkMatterHaloScaleClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((darkMatterHaloScaleClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision dynamicalTimescale(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) The characteristic dynamical timescale of a dark matter halo.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision meanDensity(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) The mean density of a dark matter halo.

double precision meanDensityGrowthRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) The growth rate of the mean density of a dark matter halo.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision virialRadius(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) The virial radius of a dark matter halo.

double precision virialRadiusGradientLogarithmicMass(\textcolor{red}{\textless type((treeNode))\textgre node\arginout) The logarithmic gradient of virial radius of a dark matter halo with halo mass at fixed epoch.

double precision virialRadiusGrowthRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) The growth rate of the virial radius of a dark matter halo.

double precision virialTemperature(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) The virial temperature of a dark matter halo

double precision virialVelocity(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) The virial velocity of a dark matter halo

double precision virialVelocityGrowthRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) The growth rate of the virial velocity of a dark matter halo.

darkMatterHaloScaleVirialDensityContrastDefinition

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`<void> calculationReset(<type(table)> node↔)` Reset memoized calculations.

`void deepCopy(\textcolor{red}{\textless class((darkMatterHaloScaleClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision dynamicalTimescale(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` The characteristic dynamical timescale of a dark matter halo.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision meanDensity(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` The mean density of a dark matter halo.

`double precision meanDensityGrowthRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` The growth rate of the mean density of a dark matter halo.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision virialRadius(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` The virial radius of a dark matter halo.

`double precision virialRadiusGradientLogarithmicMass(\textcolor{red}{\textless type((treeNode))\textgre node\arginout)` The logarithmic gradient of virial radius of a dark matter halo with halo mass at fixed epoch.

`double precision virialRadiusGrowthRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` The growth rate of the virial radius of a dark matter halo.

double precision virialTemperature(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) The virial temperature of a dark matter halo

double precision virialVelocity(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) The virial velocity of a dark matter halo

double precision virialVelocityGrowthRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) The growth rate of the virial velocity of a dark matter halo.

darkMatterParticleCDM

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((darkMatterParticleClass))\textgreater} destination\argin Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

darkMatterParticleClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((darkMatterParticleClass))\textgreater} destination\argin Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

darkMatterParticleWDMThermal

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((darkMatterParticleClass))\textgreater} destination\argi` Perform a deep copy of the object.

`<double> degreesOfFreedomEffective()` Return the effective number of degrees of freedom of the thermal wark dark matter particle.

`<double> degreesOfFreedomEffectiveDecoupling()` Return the effective number of relativisitc degrees of freedom in the universe at the time at which the thermal wark dark matter particle decoupled.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`<double> mass()` Return the mass of the thermal wark dark matter particle in units of keV.

`type(varying_string) objectType()` Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

darkMatterProfileAdiabaticGnedin2004

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

<void> calculationReset(<type(table)> node↔) Reset memoized calculations.

double precision circularVelocity(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} radius\argin) Returns the circular velocity (in km/s) in the dark matter profile of node at the given radius (given in units of Mpc).

double precision circularVelocityMaximum(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the maximum circular velocity (in km/s) in the dark matter profile of node.

double precision circularVelocityMaximumNumerical(<type((treeNode))> node↔) Returns the maximum circular velocity (in km/s) in the dark matter profile of node.

double precision circularVelocityNumerical(<type((treeNode))> node↔,<double precision> radius
argin) Returns the circular velocity (in km/s) in the dark matter profile of node at the given radius (given in units of Mpc).

<void> computeFactors(<type(treeNode)> node↔, <double> radius→, <logical> [computeGradientFactors]→) Compute factors needed for solving adiabatic contraction.

void deepCopy(\textcolor{red}{\textless class((darkMatterProfileClass))\textgreater} destination\argino Perform a deep copy of the object.

double precision density(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcol double precision\textgreater} radius\argin) Returns the density (in $M_\odot$ Mpc^{-3}) in the dark matter profile of node at the given radius (given in units of Mpc).

double precision densityLogSlope(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} radius\argin) Returns the logarithmic slope of the density profile in the dark matter profile of node at the given radius (given in units of Mpc).

`double precision densityLogSlopeNumerical(<type((treeNode))> node↔,<double precision> radius→)` Returns the logarithmic slope of the density in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision enclosedMass(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\te double precision\textgreater} radius\argin)` Returns the enclosed mass (in $M_\odot$) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`double precision enclosedMassDifferenceNumerical(<type((treeNode))> node↔, <double precision> radiusLower→, <double precision> radiusUpper→)` Returns the enclosed mass difference (in $M_\odot$) in the dark matter profile of `node` between the given `radiusLower` and `radiusUpper` (given in units of Mpc) using a numerical calculation.

`double precision enclosedMassNumerical(<type((treeNode))> node↔,<double precision> radius→)` Returns the enclosed mass (in $M_\odot$) in the dark matter profile of `node` at the given `radius` (given in units of Mpc) using a numerical calculation.

`double precision energy(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Return the total energy for the given `node` in units of $M_\odot$ km^2 s^{-2}.

`double precision energyGrowthRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Return the rate of change of total energy for the given `node` in units of $M_\odot$ km^2 s^{-2} Gyr^{-1}.

`double precision energyGrowthRateNumerical(<type((treeNode))> node↔)` Return the rate of growth of the energy of the dark matter density profile.

`double precision energyNumerical(<type((treeNode))> node↔)` Return the energy of the dark matter density profile.

`double precision freefallRadius(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} time\argin)` Returns the freefall radius (in Mpc) corresponding to the given `time` (in Gyr) in `node`.

`double precision freeFallRadiusIncreaseRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} time\argin)` Returns the rate of increase of the freefall radius (in Mpc/Gyr) corresponding to the given `time` (in Gyr) in `node`.

`double precision freefallRadiusIncreaseRateNumerical(<type((treeNode))> node↔,<double precision> time→)` Returns the rate of increase of the freefall radius in the dark matter density profile at the specified `time` (given in Gyr).

`double precision freefallRadiusNumerical(<type((treeNode))> node↔,<double precision> time→)` Returns the freefall radius in the dark matter density profile at the specified `time` (given in Gyr).

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision kSpace(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolo double precision\textgreater} wavenumber\argin)` Returns the normalized Fourier space density profile of the dark matter profile of `node` at the given `waveNumber` (given in units of Mpc^{-1}).

`double precision kSpaceNumerical(<type((treeNode))> node↔, <double> waveNumber→)` Returns the Fourier transform of the adiabaticGnedin2004 density profile at the specified `waveNumber` (given in Mpc^{-1}).

`type(varying_string) objectType()` Return the type of the object.

`double precision potential(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textc double precision\textgreater} radius\argin,\textcolor{red}{\textless integer\textgreater} status\argout)` Returns the gravitational potential (in $(\mathrm{km/s})^2$) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`double precision potentialDifferenceNumerical(<type((treeNode))> node↔, <double precision> radiusLower→, <double precision> radiusUpper→)` Returns the gravitational potential difference (in $(\mathrm{km/s})^2$) in the dark matter profile of `node` between the given `radiusLower` and `radiusUpper` (given in units of Mpc) using a numerical calculation.

`double precision potentialNumerical(<type((treeNode))> node↔,<double precision> radius→,<integer> status←)` Returns the gravitational potential (in $(\mathrm{km/s})^2$) in the dark matter profile of `node` at the given `radius` (given in units of Mpc) using a numerical calculation.

`double precision radialMoment(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\te double precision\textgreater} moment\argin,\textcolor{red}{\textless double precision\textgreater} radiusMinimum\argin,\textcolor{red}{\textless double precision\textgreater} radiusMaximum\argin)` Returns the $\mathtt{m}^{\mathrm{th}}$ radial moment of the dark matter profile of `node` optionally between the given `radiusMinimum` and `radiusMaximum` (given in units of Mpc).

`double precision radialMomentNumerical(<type((treeNode))> node↔, <double> moment→, <double> radiusMinimum→, <double> radiusMaximum→)` Returns the radial moment of the density in the dark matter profile of `node` between the given `radiusMinimum` and `radiusMaximum` (given in units of Mpc).

`double precision radialVelocityDispersion(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} radius\argin)` Returns the radial velocity dispersion (in km/s) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`double precision radialVelocityDispersionNumerical(<type((treeNode))> node↔,<double precision> radius argin)` Returns the radial velocity dispersion (in km/s) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`double precision radiusEnclosingDensity(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} density\argin)` Returns the radius (in Mpc) enclosing a given density threshold (in $M_\odot\mathrm{Mpc}^{-3}$) in the dark matter profile of `node`.

`double precision radiusEnclosingDensityNumerical(<type((treeNode))> node↔,<double precision> density→)` Returns the radius (in Mpc) in the dark matter profile of `node` which encloses the given `density` (given in units of $M_\odot/\mathrm{Mpc}^{-3}$).

`double precision radiusEnclosingMass(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} mass\argin)` Returns the radius (in Mpc) enclosing a given mass (in $M_\odot$) in the dark matter profile of `node`.

`double precision radiusEnclosingMassNumerical(<type((treeNode))> node↔,<double precision> mass→)` Returns the radius (in Mpc) in the dark matter profile of `node` which encloses the given `mass` (given in units of $M_\odot$).

`double precision radiusFromSpecificAngularMomentum(\textcolor{red}{\textless type((treeNode))\textgreat node\arginout,\textcolor{red}{\textless double precision\textgreater} specificAngularMomentum\argin` Returns the radius (in Mpc) in the dark matter profile of `node` at which the specific angular momentum of a circular orbit equals `specificAngularMomentum` (specified in units of km s^{-1} Mpc.

`double precision radiusFromSpecificAngularMomentumNumerical(<type((treeNode))> node↔,<double precision> specificAngularMomentum→)` Returns the radius (in Mpc) in `node` at which a circular orbit has the given `specificAngularMomentum` (given in units of km s^{-1} Mpc).

`<double> radiusInitial(<type(treeNode)> node↔, <double> radius→)` Compute the initial radius in the dark matter profile.

`<double> radiusInitialDerivative(<type(treeNode)> node↔, <double> radius→)` Compute the derivative of the initial radius in the dark matter profile.

`<double> radiusOrbitalMean(<double> radius→)` Compute the orbit-averaged radius for dark matter.

`<double> radiusOrbitalMeanDerivative(<double> radius→)` Compute the derivative of the orbit-averaged radius for dark matter.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`double precision rotationNormalization(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Returns the relation between specific angular momentum and rotation velocity (assuming a rotation velocity that is constant in radius) for the given `node`. Specifically, the normalization, A, returned is such that $V_{\rm rot} = AJ/M$

`double precision rotationNormalizationNumerical(<type((treeNode))> node↔)` Return the normalization of the rotation velocity vs. specific angular momentum relation.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

darkMatterProfileClass

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`double precision circularVelocity(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} radius\argin)` Returns the circular velocity (in km/s) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`double precision circularVelocityMaximum(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Returns the maximum circular velocity (in km/s) in the dark matter profile of `node`.

`double precision circularVelocityMaximumNumerical(<type((treeNode))> node↔)` Returns the maximum circular velocity (in km/s) in the dark matter profile of `node`.

`double precision circularVelocityNumerical(<type((treeNode))> node↔,<double precision> radius argin)` Returns the circular velocity (in km/s) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`void deepCopy(\textcolor{red}{\textless class((darkMatterProfileClass))\textgreater} destination\argino` Perform a deep copy of the object.

`double precision density(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcol double precision\textgreater} radius\argin)` Returns the density (in $M_\odot$ Mpc^{-3}) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`double precision densityLogSlope(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} radius\argin)` Returns the logarithmic slope of the density profile in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`double precision densityLogSlopeNumerical(<type((treeNode))> node↔,<double precision> radius→)` Returns the logarithmic slope of the density in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision enclosedMass(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\te double precision\textgreater} radius\argin)` Returns the enclosed mass (in $M_\odot$) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`double precision enclosedMassDifferenceNumerical(<type((treeNode))> node↔, <double precision> radiusLower→, <double precision> radiusUpper→)` Returns the enclosed mass difference (in $M_\odot$) in the dark matter profile of `node` between the given `radiusLower` and `radiusUpper` (given in units of Mpc) using a numerical calculation.

`double precision enclosedMassNumerical(<type((treeNode))> node↔,<double precision> radius→)` Returns the enclosed mass (in $M_\odot$) in the dark matter profile of `node` at the given `radius` (given in units of Mpc) using a numerical calculation.

`double precision energy(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Return the total energy for the given `node` in units of $M_\odot$ km^2 s^{-2}.

`double precision energyGrowthRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Return the rate of change of total energy for the given `node` in units of $M_\odot$ km^2 s^{-2} Gyr^{-1}.

`double precision energyGrowthRateNumerical(<type((treeNode))> node↔)` Return the rate of growth of the energy of the dark matter density profile.

`double precision energyNumerical(<type((treeNode))> node↔)` Return the energy of the dark matter density profile.

`double precision freefallRadius(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} time\argin)` Returns the freefall radius (in Mpc) corresponding to the given `time` (in Gyr) in `node`.

`double precision freeFallRadiusIncreaseRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} time\argin)` Returns the rate of increase of the freefall radius (in Mpc/Gyr) corresponding to the given `time` (in Gyr) in `node`.

`double precision freefallRadiusIncreaseRateNumerical(<type((treeNode))> node↔,<double precision> time→)` Returns the rate of increase of the freefall radius in the dark matter density profile at the specified `time` (given in Gyr).

`double precision freefallRadiusNumerical(<type((treeNode))> node↔,<double precision> time→)` Returns the freefall radius in the dark matter density profile at the specified `time` (given in Gyr).

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision kSpace(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolo double precision\textgreater} wavenumber\argin)` Returns the normalized Fourier space density profile of the dark matter profile of `node` at the given `waveNumber` (given in units of Mpc^{-1}).

`double precision kSpaceNumerical(<type((treeNode))> node↔, <double> waveNumber→)` Returns the Fourier transform of the adiabaticGnedin2004 density profile at the specified `waveNumber` (given in Mpc^{-1}).

`type(varying_string) objectType()` Return the type of the object.

`double precision potential(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textc double precision\textgreater} radius\argin,\textcolor{red}{\textless integer\textgreater} status\argout)` Returns the gravitational potential (in (km/s)^2) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

double precision potentialDifferenceNumerical(<type((treeNode))> node↔, <double precision> radiusLower→, <double precision> radiusUpper→) Returns the gravitational potential difference (in $(\mathrm{km/s})^2$) in the dark matter profile of node between the given radiusLower and radiusUpper (given in units of Mpc) using a numerical calculation.

double precision potentialNumerical(<type((treeNode))> node↔,<double precision> radius→,<integer> status←) Returns the gravitational potential (in $(\mathrm{km/s})^2$) in the dark matter profile of node at the given radius (given in units of Mpc) using a numerical calculation.

double precision radialMoment(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\te double precision\textgreater} moment\argin,\textcolor{red}{\textless double precision\textgreater} radiusMinimum\argin,\textcolor{red}{\textless double precision\textgreater} radiusMaximum\argin) Returns the $\mathtt{m}^{\mathrm{th}}$ radial moment of the dark matter profile of node optionally between the given radiusMinimum and radiusMaximum (given in units of Mpc).

double precision radialMomentNumerical(<type((treeNode))> node↔, <double> moment→, <double> radiusMinimum→, <double> radiusMaximum→) Returns the radial moment of the density in the dark matter profile of node between the given radiusMinimum and radiusMaximum (given in units of Mpc).

double precision radialVelocityDispersion(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} radius\argin) Returns the radial velocity dispersion (in km/s) in the dark matter profile of node at the given radius (given in units of Mpc).

double precision radialVelocityDispersionNumerical(<type((treeNode))> node↔,<double precision> radius
argin) Returns the radial velocity dispersion (in km/s) in the dark matter profile of node at the given radius (given in units of Mpc).

double precision radiusEnclosingDensity(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} density\argin) Returns the radius (in Mpc) enclosing a given density threshold (in $M_\odot\mathrm{Mpc}^{-3}$) in the dark matter profile of node.

double precision radiusEnclosingDensityNumerical(<type((treeNode))> node↔,<double precision> density→) Returns the radius (in Mpc) in the dark matter profile of node which encloses the given density (given in units of $M_\odot/\mathrm{Mpc}^{-3}$).

double precision radiusEnclosingMass(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} mass\argin) Returns the radius (in Mpc) enclosing a given mass (in $M_\odot$) in the dark matter profile of node.

double precision radiusEnclosingMassNumerical(<type((treeNode))> node↔,<double precision> mass→) Returns the radius (in Mpc) in the dark matter profile of node which encloses the given mass (given in units of $M_\odot$).

double precision radiusFromSpecificAngularMomentum(\textcolor{red}{\textless type((treeNode))\textgreat node\arginout,\textcolor{red}{\textless double precision\textgreater} specificAngularMomentum\argin Returns the radius (in Mpc) in the dark matter profile of node at which the specific angular momentum of a circular orbit equals specificAngularMomentum (specified in units of km s^{-1} Mpc.

`double precision radiusFromSpecificAngularMomentumNumerical(<type((treeNode))> node↔,<double precision> specificAngularMomentum→)` Returns the radius (in Mpc) in `node` at which a circular orbit has the given `specificAngularMomentum` (given in units of km s^{-1} Mpc).

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`double precision rotationNormalization(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Returns the relation between specific angular momentum and rotation velocity (assuming a rotation velocity that is constant in radius) for the given `node`. Specifically, the normalization, A, returned is such that $V_{\rm rot} = AJ/M$

`double precision rotationNormalizationNumerical(<type((treeNode))> node↔)` Return the normalization of the rotation velocity vs. specific angular momentum relation.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

darkMatterProfileConcentrationBullock2001

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`double precision concentration(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Returns the concentration parameter for the given `node`.

`double precision concentrationMean(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Returns the mean concentration parameter for a `node` of the given mass.

`class(darkMatterProfileDMOClass) darkMatterProfileDMODefinition()` Returns a `darkMatterProfileDMO` object describing the dark matter density profile used to define this concentration.

`void deepCopy(\textcolor{red}{\textless class((darkMatterProfileConcentrationClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`class(virialDensityContrastClass) densityContrastDefinition()` Returns a `virialDensityContrast` object describing the virial density contrast used to define this concentration.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

darkMatterProfileConcentrationClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

double precision concentration(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the concentration parameter for the given node.

double precision concentrationMean(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the mean concentration parameter for a node of the given mass.

class(darkMatterProfileDMOClass) darkMatterProfileDMODefinition() Returns a darkMatterProfileDMO object describing the dark matter density profile used to define this concentration.

void deepCopy(\textcolor{red}{\textless class((darkMatterProfileConcentrationClass))\textgreater} destination\arginout) Perform a deep copy of the object.

class(virialDensityContrastClass) densityContrastDefinition() Returns a virialDensityContrast object describing the virial density contrast used to define this concentration.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

darkMatterProfileConcentrationCorrea2015

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

double precision concentration(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the concentration parameter for the given node.

double precision concentrationMean(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the mean concentration parameter for a node of the given mass.

class(darkMatterProfileDMOClass) darkMatterProfileDMODefinition() Returns a darkMatterProfileDMO object describing the dark matter density profile used to define this concentration.

void deepCopy(\textcolor{red}{\textless class((darkMatterProfileConcentrationClass))\textgreater} destination\arginout) Perform a deep copy of the object.

class(virialDensityContrastClass) densityContrastDefinition() Returns a virialDensityContrast object describing the virial density contrast used to define this concentration.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

darkMatterProfileConcentrationDiemerKravtsov2014

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

double precision concentration(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the concentration parameter for the given node.

double precision concentrationMean(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the mean concentration parameter for a node of the given mass.

class(darkMatterProfileDMOClass) darkMatterProfileDMODefinition() Returns a darkMatterProfileDMO object describing the dark matter density profile used to define this concentration.

void deepCopy(\textcolor{red}{\textless class((darkMatterProfileConcentrationClass))\textgreater} destination\arginout) Perform a deep copy of the object.

class(virialDensityContrastClass) densityContrastDefinition() Returns a virialDensityContrast object describing the virial density contrast used to define this concentration.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

darkMatterProfileConcentrationDuttonMaccio2014

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`double precision concentration(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Returns the concentration parameter for the given `node`.

`double precision concentrationMean(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Returns the mean concentration parameter for a `node` of the given mass.

`class(darkMatterProfileDMOClass) darkMatterProfileDMODefinition()` Returns a `darkMatterProfileDMO` object describing the dark matter density profile used to define this concentration.

`void deepCopy(\textcolor{red}{\textless class((darkMatterProfileConcentrationClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`<void> definitions()` Establish definitions for virial density contrast and dark matter halo profile.

`class(virialDensityContrastClass) densityContrastDefinition()` Returns a `virialDensityContrast` object describing the virial density contrast used to define this concentration.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

darkMatterProfileConcentrationGao2008

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

double precision concentration(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the concentration parameter for the given node.

double precision concentrationMean(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the mean concentration parameter for a node of the given mass.

class(darkMatterProfileDMOClass) darkMatterProfileDMODefinition() Returns a darkMatterProfileDMO object describing the dark matter density profile used to define this concentration.

void deepCopy(\textcolor{red}{\textless class((darkMatterProfileConcentrationClass))\textgreater} destination\arginout) Perform a deep copy of the object.

class(virialDensityContrastClass) densityContrastDefinition() Returns a virialDensityContrast object describing the virial density contrast used to define this concentration.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

darkMatterProfileConcentrationKlypin2015

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

double precision concentration(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the concentration parameter for the given node.

double precision concentrationMean(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the mean concentration parameter for a node of the given mass.

class(darkMatterProfileDMOClass) darkMatterProfileDMODefinition() Returns a darkMatterProfileDMO object describing the dark matter density profile used to define this concentration.

void deepCopy(\textcolor{red}{\textless class((darkMatterProfileConcentrationClass))\textgreater} destination\arginout) Perform a deep copy of the object.

class(virialDensityContrastClass) densityContrastDefinition() Returns a virialDensityContrast object describing the virial density contrast used to define this concentration.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

darkMatterProfileConcentrationLudlow2016Fit

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

double precision concentration(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the concentration parameter for the given node.

double precision concentrationMean(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the mean concentration parameter for a node of the given mass.

class(darkMatterProfileDMOClass) darkMatterProfileDMODefinition() Returns a darkMatterProfileDMO object describing the dark matter density profile used to define this concentration.

void deepCopy(\textcolor{red}{\textless class((darkMatterProfileConcentrationClass))\textgreater} destination\arginout) Perform a deep copy of the object.

class(virialDensityContrastClass) densityContrastDefinition() Returns a virialDensityContrast object describing the virial density contrast used to define this concentration.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

darkMatterProfileConcentrationMunozCuartas2011

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

double precision concentration(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the concentration parameter for the given node.

double precision concentrationMean(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the mean concentration parameter for a node of the given mass.

class(darkMatterProfileDMOClass) darkMatterProfileDMODefinition() Returns a darkMatterProfileDMO object describing the dark matter density profile used to define this concentration.

void deepCopy(\textcolor{red}{\textless class((darkMatterProfileConcentrationClass))\textgreater} destination\arginout) Perform a deep copy of the object.

class(virialDensityContrastClass) densityContrastDefinition() Returns a virialDensityContrast object describing the virial density contrast used to define this concentration.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

darkMatterProfileConcentrationNFW1996

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

double precision concentration(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the concentration parameter for the given node.

double precision concentrationMean(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the mean concentration parameter for a node of the given mass.

class(darkMatterProfileDMOClass) darkMatterProfileDMODefinition() Returns a darkMatterProfileDMO object describing the dark matter density profile used to define this concentration.

void deepCopy(\textcolor{red}{\textless class((darkMatterProfileConcentrationClass))\textgreater} destination\arginout) Perform a deep copy of the object.

class(virialDensityContrastClass) densityContrastDefinition() Returns a virialDensityContrast object describing the virial density contrast used to define this concentration.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

darkMatterProfileConcentrationPrada2011

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

double precision concentration(\textcolor{red}{\textless type((treeNode))\textgreater}
node\arginout) Returns the concentration parameter for the given node.

double precision concentrationMean(\textcolor{red}{\textless type((treeNode))\textgreater}
node\arginout) Returns the mean concentration parameter for a node of the given mass.

class(darkMatterProfileDMOClass) darkMatterProfileDMODefinition() Returns a darkMatterProfileDMO object describing the dark matter density profile used to define this concentration.

void deepCopy(\textcolor{red}{\textless class((darkMatterProfileConcentrationClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

class(virialDensityContrastClass) densityContrastDefinition() Returns a virialDensityContrast object describing the virial density contrast used to define this concentration.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

darkMatterProfileConcentrationSchneider2015

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

double precision concentration(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the concentration parameter for the given node.

double precision concentrationMean(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the mean concentration parameter for a node of the given mass.

class(darkMatterProfileDMOClass) darkMatterProfileDMODefinition() Returns a darkMatterProfileDMO object describing the dark matter density profile used to define this concentration.

void deepCopy(\textcolor{red}{\textless class((darkMatterProfileConcentrationClass))\textgreater} destination\arginout) Perform a deep copy of the object.

class(virialDensityContrastClass) densityContrastDefinition() Returns a virialDensityContrast object describing the virial density contrast used to define this concentration.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

darkMatterProfileConcentrationWDM

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

double precision concentration(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the concentration parameter for the given node.

double precision concentrationMean(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the mean concentration parameter for a node of the given mass.

class(darkMatterProfileDMOClass) darkMatterProfileDMODefinition() Returns a darkMatterProfileDMO object describing the dark matter density profile used to define this concentration.

void deepCopy(\textcolor{red}{\textless class((darkMatterProfileConcentrationClass))\textgreater} destination\arginout) Perform a deep copy of the object.

class(virialDensityContrastClass) densityContrastDefinition() Returns a virialDensityContrast object describing the virial density contrast used to define this concentration.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

darkMatterProfileConcentrationZhao2009

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`double precision concentration(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Returns the concentration parameter for the given `node`.

`double precision concentrationMean(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Returns the mean concentration parameter for a `node` of the given mass.

`class(darkMatterProfileDMOClass) darkMatterProfileDMODefinition()` Returns a `darkMatterProfileDMO` object describing the dark matter density profile used to define this concentration.

`void deepCopy(\textcolor{red}{\textless class((darkMatterProfileConcentrationClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`class(virialDensityContrastClass) densityContrastDefinition()` Returns a `virialDensityContrast` object describing the virial density contrast used to define this concentration.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

darkMatterProfileDarkMatterOnly

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`double precision circularVelocity(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} radius\argin)` Returns the circular velocity (in km/s) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`double precision circularVelocityMaximum(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Returns the maximum circular velocity (in km/s) in the dark matter profile of `node`.

`double precision circularVelocityMaximumNumerical(<type((treeNode))> node↔)` Returns the maximum circular velocity (in km/s) in the dark matter profile of `node`.

`double precision circularVelocityNumerical(<type((treeNode))> node↔,<double precision> radius argin)` Returns the circular velocity (in km/s) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`void deepCopy(\textcolor{red}{\textless class((darkMatterProfileClass))\textgreater} destination\argino` Perform a deep copy of the object.

`double precision density(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcol double precision\textgreater} radius\argin)` Returns the density (in $M_\odot$ Mpc^{-3}) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`double precision densityLogSlope(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} radius\argin)` Returns the logarithmic slope of the density profile in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`double precision densityLogSlopeNumerical(<type((treeNode))> node↔,<double precision> radius→)` Returns the logarithmic slope of the density in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision enclosedMass(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\te double precision\textgreater} radius\argin)` Returns the enclosed mass (in $M_\odot$) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`double precision enclosedMassDifferenceNumerical(<type((treeNode))> node↔, <double precision> radiusLower→, <double precision> radiusUpper→)` Returns the enclosed mass difference (in $M_\odot$) in the dark matter profile of `node` between the given `radiusLower` and `radiusUpper` (given in units of Mpc) using a numerical calculation.

`double precision enclosedMassNumerical(<type((treeNode))> node↔,<double precision> radius→)` Returns the enclosed mass (in $M_\odot$) in the dark matter profile of `node` at the given `radius` (given in units of Mpc) using a numerical calculation.

`double precision energy(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Return the total energy for the given `node` in units of $M_\odot$ km^2 s^{-2}.

`double precision energyGrowthRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Return the rate of change of total energy for the given `node` in units of $M_\odot$ km^2 s^{-2} Gyr^{-1}.

`double precision energyGrowthRateNumerical(<type((treeNode))> node↔)` Return the rate of growth of the energy of the dark matter density profile.

`double precision energyNumerical(<type((treeNode))> node↔)` Return the energy of the dark matter density profile.

`double precision freefallRadius(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} time\argin)` Returns the freefall radius (in Mpc) corresponding to the given `time` (in Gyr) in `node`.

`double precision freeFallRadiusIncreaseRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} time\argin)` Returns the rate of increase of the freefall radius (in Mpc/Gyr) corresponding to the given `time` (in Gyr) in `node`.

`double precision freefallRadiusIncreaseRateNumerical(<type((treeNode))> node↔,<double precision> time→)` Returns the rate of increase of the freefall radius in the dark matter density profile at the specified `time` (given in Gyr).

`double precision freefallRadiusNumerical(<type((treeNode))> node↔,<double precision> time→)` Returns the freefall radius in the dark matter density profile at the specified `time` (given in Gyr).

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision kSpace(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolo double precision\textgreater} wavenumber\argin)` Returns the normalized Fourier space density profile of the dark matter profile of `node` at the given `waveNumber` (given in units of Mpc^{-1}).

`double precision kSpaceNumerical(<type((treeNode))> node↔, <double> waveNumber→)` Returns the Fourier transform of the adiabaticGnedin2004 density profile at the specified `waveNumber` (given in Mpc^{-1}).

`type(varying_string) objectType()` Return the type of the object.

`double precision potential(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textc double precision\textgreater} radius\argin,\textcolor{red}{\textless integer\textgreater} status\argout)` Returns the gravitational potential (in (km/s)^2) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`double precision potentialDifferenceNumerical(<type((treeNode))> node↔, <double precision> radiusLower→, <double precision> radiusUpper→)` Returns the gravitational potential difference (in (km/s)^2) in the dark matter profile of `node` between the given `radiusLower` and `radiusUpper` (given in units of Mpc) using a numerical calculation.

`double precision potentialNumerical(<type((treeNode))> node↔,<double precision> radius→,<integer> status←)` Returns the gravitational potential (in (km/s)^2) in the dark matter profile of `node` at the given `radius` (given in units of Mpc) using a numerical calculation.

`double precision radialMoment(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\te double precision\textgreater} moment\argin,\textcolor{red}{\textless double precision\textgreater} radiusMinimum\argin,\textcolor{red}{\textless double precision\textgreater} radiusMaximum\argin)` Returns the $\mathtt{m}^{\mathrm{th}}$ radial moment of the dark matter profile of `node` optionally between the given `radiusMinimum` and `radiusMaximum` (given in units of Mpc).

`double precision radialMomentNumerical(<type((treeNode))> node↔, <double> moment→, <double> radiusMinimum→, <double> radiusMaximum→)` Returns the radial moment of the density in the dark matter profile of `node` between the given `radiusMinimum` and `radiusMaximum` (given in units of Mpc).

`double precision radialVelocityDispersion(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} radius\argin)` Returns the radial velocity dispersion (in km/s) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`double precision radialVelocityDispersionNumerical(<type((treeNode))> node↔,<double precision> radius argin)` Returns the radial velocity dispersion (in km/s) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`double precision radiusEnclosingDensity(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} density\argin)` Returns the radius (in Mpc) enclosing a given density threshold (in $M_\odot\mathrm{Mpc}^{-3}$) in the dark matter profile of `node`.

`double precision radiusEnclosingDensityNumerical(<type((treeNode))> node↔,<double precision> density→)` Returns the radius (in Mpc) in the dark matter profile of `node` which encloses the given `density` (given in units of $M_\odot/\mathrm{Mpc}^{-3}$).

`double precision radiusEnclosingMass(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} mass\argin)` Returns the radius (in Mpc) enclosing a given mass (in $M_\odot$) in the dark matter profile of `node`.

`double precision radiusEnclosingMassNumerical(<type((treeNode))> node↔,<double precision> mass→)` Returns the radius (in Mpc) in the dark matter profile of `node` which encloses the given `mass` (given in units of $M_\odot$).

`double precision radiusFromSpecificAngularMomentum(\textcolor{red}{\textless type((treeNode))\textgreat node\arginout,\textcolor{red}{\textless double precision\textgreater} specificAngularMomentum\argin` Returns the radius (in Mpc) in the dark matter profile of `node` at which the specific angular momentum of a circular orbit equals `specificAngularMomentum` (specified in units of km s^{-1} Mpc.

`double precision radiusFromSpecificAngularMomentumNumerical(<type((treeNode))> node↔,<double precision> specificAngularMomentum→)` Returns the radius (in Mpc) in `node` at which a circular orbit has the given `specificAngularMomentum` (given in units of km s^{-1} Mpc).

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`double precision rotationNormalization(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Returns the relation between specific angular momentum and rotation velocity (assuming a rotation velocity that is constant in radius) for the given `node`. Specifically, the normalization, A, returned is such that $V_{\rm rot} = AJ/M$

`double precision rotationNormalizationNumerical(<type((treeNode))> node↔)` Return the normalization of the rotation velocity vs. specific angular momentum relation.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

darkMatterProfileDMOBurkert

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`<double> angularMomentumScaleFree(<double> concentration→)` Returns the total angular momentum in an Burkert dark matter profile with given `concentration`.

`void autoHook()` Insert any event hooks required by this object.

`<void> calculationReset(<type(table)> node↔)` Reset memoized calculations.

`double precision circularVelocity(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} radius\argin)` Returns the circular velocity (in km/s) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`double precision circularVelocityMaximum(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Returns the maximum circular velocity (in km/s) in the dark matter profile of `node`.

`double precision circularVelocityMaximumNumerical(<type((treeNode))> node↔)` Returns the maximum circular velocity (in km/s) in the dark matter profile of `node`.

`double precision circularVelocityNumerical(<type((treeNode))> node↔,<double precision> radius`
`argin)` Returns the circular velocity (in km/s) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`void deepCopy(\textcolor{red}{\textless class((darkMatterProfileDMOClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`double precision density(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcol double precision\textgreater} radius\argin)` Returns the density (in $M_\odot$ Mpc^{-3}) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`double precision densityLogSlope(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} radius\argin)` Returns the logarithmic slope of the density profile in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`double precision densityLogSlopeNumerical(<type((treeNode))> node↔,<double precision> radius→)` Returns the logarithmic slope of the density in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`<double> densityScaleFree(<double> radius→, <double> concentration→)` Returns the density (in units such that the virial mass and scale length are unity) in a Burkert dark matter profile with given `concentration` at the given `radius` (given in units of the scale radius).

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision enclosedMass(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\te double precision\textgreater} radius\argin)` Returns the enclosed mass (in $M_\odot$) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`double precision enclosedMassDifferenceNumerical(<type((treeNode))> node↔, <double precision> radiusLower→, <double precision> radiusUpper→)` Returns the enclosed mass difference (in $M_\odot$) in the dark matter profile of `node` between the given `radiusLower` and `radiusUpper` (given in units of Mpc) using a numerical calculation.

`double precision enclosedMassNumerical(<type((treeNode))> node↔,<double precision> radius→)` Returns the enclosed mass (in $M_\odot$) in the dark matter profile of `node` at the given `radius` (given in units of Mpc) using a numerical calculation.

`<double> enclosedMassScaleFree(<double> radius→, <double> concentration→)` Returns the enclosed mass (in units of the virial mass) in a Burkert dark matter profile with given `concentration` at the given `radius` (given in units of the scale radius).

`double precision energy(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Return the total energy for the given `node` in units of $M_\odot$ km^2 s^{-2}.

`double precision energyGrowthRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Returns the rate of chance of the total energy of `node` in units of $M_\odot$ km^2 s^{-2} Gyr^{-1}.

`double precision energyGrowthRateNumerical(<type((treeNode))> node↔)` Return the rate of growth of the energy of the dark matter density profile.

`double precision energyNumerical(<type((treeNode))> node↔)` Return the energy of the dark matter density profile.

`double precision freefallRadius(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} time\argin)` Returns the freefall radius (in Mpc) corresponding to the given `time` (in Gyr) in `node`.

`double precision freeFallRadiusIncreaseRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} time\argin)` Returns the rate of increase of the freefall radius (in Mpc/Gyr) corresponding to the given `time` (in Gyr) in `node`.

double precision freefallRadiusIncreaseRateNumerical(<type((treeNode))> node↔,<double precision> time→) Returns the rate of increase of the freefall radius in the dark matter density profile at the specified time (given in Gyr).

double precision freefallRadiusNumerical(<type((treeNode))> node↔,<double precision> time→) Returns the freefall radius in the dark matter density profile at the specified time (given in Gyr).

<void> freefallTabulate(<double> freefallTimeScaleFree→) Tabulates the freefall time vs. freefall radius for Burkert halos.

<double> freefallTimeScaleFree(<double> radius→) Compute the freefall time in a scale-free Burkert halo.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<void> inverseAngularMomentum(<double> specificAngularMomentum→) Tabulates the specific angular momentum vs. radius in an Burkert profile for rapid inversion.

<logical> isDefault() Return true if this is the default object of this class.

double precision kSpace(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolo double precision\textgreater} wavenumber\argin) Returns the normalized Fourier space density profile of the dark matter profile of node at the given waveNumber (given in units of Mpc^{-1}).

double precision kSpaceNumerical(<type((treeNode))> node↔, <double> waveNumber→) Returns the Fourier transform of the adiabaticGnedin2004 density profile at the specified waveNumber (given in Mpc^{-1}).

type(varying_string) objectType() Return the type of the object.

double precision potential(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textc double precision\textgreater} radius\argin,\textcolor{red}{\textless integer\textgreater} status\argout) Returns the gravitational potential (in $(\mathrm{km/s})^2$) in the dark matter profile of node at the given radius (given in units of Mpc).

double precision potentialDifferenceNumerical(<type((treeNode))> node↔, <double precision> radiusLower→, <double precision> radiusUpper→) Returns the gravitational potential difference (in $(\mathrm{km/s})^2$) in the dark matter profile of node between the given radiusLower and radiusUpper (given in units of Mpc) using a numerical calculation.

double precision potentialNumerical(<type((treeNode))> node↔,<double precision> radius→,<integer> status←) Returns the gravitational potential (in $(\mathrm{km/s})^2$) in the dark matter profile of node at the given radius (given in units of Mpc) using a numerical calculation.

<double> profileEnergy(<double> concentration→) Computes the total energy of an Burkert profile halo of given concentration.

double precision radialMoment(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\te double precision\textgreater} moment\argin,\textcolor{red}{\textless double precision\textgreater} radiusMinimum\argin,\textcolor{red}{\textless double precision\textgreater} radiusMaximum\argin) Returns the m^{th} radial moment of the dark matter profile of node optionally between the given radiusMinimum and radiusMaximum (given in units of Mpc).

`double precision radialMomentNumerical(<type((treeNode))> node↔, <double> moment→, <double> radiusMinimum→, <double> radiusMaximum→)` Returns the radial moment of the density in the dark matter profile of `node` between the given `radiusMinimum` and `radiusMaximum` (given in units of Mpc).

`double precision radialVelocityDispersion(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} radius\argin)` Returns the radial velocity dispersion (in km/s) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`double precision radialVelocityDispersionNumerical(<type((treeNode))> node↔,<double precision> radius argin)` Returns the radial velocity dispersion (in km/s) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`<double> radialVelocityDispersionScaleFree(<double> radius→)` Returns the scale-free radial velocity dispersion in a Burkert dark matter profile.

`<void> radialVelocityDispersionTabulate(<double> radius→)` Tabulates the radial velocity dispersion vs. radius for Burkert halos.

`double precision radiusEnclosingDensity(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} density\argin)` Returns the radius (in Mpc) enclosing a given density threshold (in $M_\odot \mathrm{Mpc}^{-3}$) in the dark matter profile of `node`.

`double precision radiusEnclosingDensityNumerical(<type((treeNode))> node↔,<double precision> density→)` Returns the radius (in Mpc) in the dark matter profile of `node` which encloses the given `density` (given in units of $M_\odot/\mathrm{Mpc}^{-3}$).

`<void> radiusEnclosingDensityTabulate(<double> densityScaleFree→→)` Tabulates the radius vs. enclosed density for Burkert halos.

`double precision radiusEnclosingMass(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} mass\argin)` Returns the radius (in Mpc) enclosing a given mass (in $M_\odot$) in the dark matter profile of `node`.

`double precision radiusEnclosingMassNumerical(<type((treeNode))> node↔,<double precision> mass→)` Returns the radius (in Mpc) in the dark matter profile of `node` which encloses the given `mass` (given in units of $M_\odot$).

`double precision radiusFromSpecificAngularMomentum(\textcolor{red}{\textless type((treeNode))\textgreat node\arginout,\textcolor{red}{\textless double precision\textgreater} specificAngularMomentum\argin` Returns the radius (in Mpc) in the dark matter profile of `node` at which the specific angular momentum of a circular orbit equals `specificAngularMomentum` (specified in units of km s^{-1} Mpc.

`double precision radiusFromSpecificAngularMomentumNumerical(<type((treeNode))> node↔,<double precision> specificAngularMomentum→)` Returns the radius (in Mpc) in `node` at which a circular orbit has the given `specificAngularMomentum` (given in units of km s^{-1} Mpc).

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

double precision rotationNormalization(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the relation between specific angular momentum and rotation velocity (assuming a rotation velocity that is constant in radius) for the given node. Specifically, the normalization, A, returned is such that $V_{\rm rot} = AJ/M$

double precision rotationNormalizationNumerical(<type((treeNode))> node↔) Return the normalization of the rotation velocity vs. specific angular momentum relation.

<double> specificAngularMomentumScaleFree(<double> radius→) Returns the specific angular momentum, normalized to unit scale length and unit velocity at the scale radius, at position radius (in units of the scale radius) in an Burkert profile.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

<void> tabulate(<double> concentration→) Tabulate properties of the Burkert halo profile which must be computed numerically.

darkMatterProfileDMOClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

double precision circularVelocity(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} radius\argin) Returns the circular velocity (in km/s) in the dark matter profile of node at the given radius (given in units of Mpc).

double precision circularVelocityMaximum(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the maximum circular velocity (in km/s) in the dark matter profile of node.

double precision circularVelocityMaximumNumerical(<type((treeNode))> node↔) Returns the maximum circular velocity (in km/s) in the dark matter profile of node.

double precision circularVelocityNumerical(<type((treeNode))> node↔,<double precision> radius
argin) Returns the circular velocity (in km/s) in the dark matter profile of node at the given radius (given in units of Mpc).

void deepCopy(\textcolor{red}{\textless class((darkMatterProfileDMOClass))\textgreater} destination\arginout) Perform a deep copy of the object.

`double precision density(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcol double precision\textgreater} radius\argin)` Returns the density (in $M_\odot$ Mpc^{-3}) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`double precision densityLogSlope(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} radius\argin)` Returns the logarithmic slope of the density profile in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`double precision densityLogSlopeNumerical(<type((treeNode))> node↔,<double precision> radius→)` Returns the logarithmic slope of the density in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision enclosedMass(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\te double precision\textgreater} radius\argin)` Returns the enclosed mass (in $M_\odot$) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`double precision enclosedMassDifferenceNumerical(<type((treeNode))> node↔, <double precision> radiusLower→, <double precision> radiusUpper→)` Returns the enclosed mass difference (in $M_\odot$) in the dark matter profile of `node` between the given `radiusLower` and `radiusUpper` (given in units of Mpc) using a numerical calculation.

`double precision enclosedMassNumerical(<type((treeNode))> node↔,<double precision> radius→)` Returns the enclosed mass (in $M_\odot$) in the dark matter profile of `node` at the given `radius` (given in units of Mpc) using a numerical calculation.

`double precision energy(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Return the total energy for the given `node` in units of $M_\odot$ km^2 s^{-2}.

`double precision energyGrowthRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Returns the rate of chance of the total energy of `node` in units of $M_\odot$ km^2 s^{-2} Gyr^{-1}.

`double precision energyGrowthRateNumerical(<type((treeNode))> node↔)` Return the rate of growth of the energy of the dark matter density profile.

`double precision energyNumerical(<type((treeNode))> node↔)` Return the energy of the dark matter density profile.

`double precision freefallRadius(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} time\argin)` Returns the freefall radius (in Mpc) corresponding to the given `time` (in Gyr) in `node`.

`double precision freeFallRadiusIncreaseRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} time\argin)` Returns the rate of increase of the freefall radius (in Mpc/Gyr) corresponding to the given `time` (in Gyr) in `node`.

`double precision freefallRadiusIncreaseRateNumerical(<type((treeNode))> node↔,<double precision> time→)` Returns the rate of increase of the freefall radius in the dark matter density profile at the specified `time` (given in Gyr).

`double precision freefallRadiusNumerical(<type((treeNode))> node↔,<double precision> time→)`
Returns the freefall radius in the dark matter density profile at the specified `time` (given in Gyr).

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges`
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision kSpace(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolo`
`double precision\textgreater} wavenumber\argin)` Returns the normalized Fourier space density profile of the dark matter profile of `node` at the given `waveNumber` (given in units of Mpc^{-1}).

`double precision kSpaceNumerical(<type((treeNode))> node↔, <double> waveNumber→)` Returns the Fourier transform of the adiabaticGnedin2004 density profile at the specified `waveNumber` (given in Mpc^{-1}).

`type(varying_string) objectType()` Return the type of the object.

`double precision potential(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textc`
`double precision\textgreater} radius\argin,\textcolor{red}{\textless integer\textgreater}`
`status\argout)` Returns the gravitational potential (in $(\mathrm{km/s})^2$) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`double precision potentialDifferenceNumerical(<type((treeNode))> node↔, <double precision>`
`radiusLower→, <double precision> radiusUpper→)` Returns the gravitational potential difference (in $(\mathrm{km/s})^2$) in the dark matter profile of `node` between the given `radiusLower` and `radiusUpper` (given in units of Mpc) using a numerical calculation.

`double precision potentialNumerical(<type((treeNode))> node↔,<double precision> radius→,<integer>`
`status←)` Returns the gravitational potential (in $(\mathrm{km/s})^2$) in the dark matter profile of `node` at the given `radius` (given in units of Mpc) using a numerical calculation.

`double precision radialMoment(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\te`
`double precision\textgreater} moment\argin,\textcolor{red}{\textless double precision\textgreater}`
`radiusMinimum\argin,\textcolor{red}{\textless double precision\textgreater} radiusMaximum\argin)`
Returns the $\mathtt{m}^{\mathrm{th}}$ radial moment of the dark matter profile of `node` optionally between the given `radiusMinimum` and `radiusMaximum` (given in units of Mpc).

`double precision radialMomentNumerical(<type((treeNode))> node↔, <double> moment→, <double>`
`radiusMinimum→, <double> radiusMaximum→)` Returns the radial moment of the density in the dark matter profile of `node` between the given `radiusMinimum` and `radiusMaximum` (given in units of Mpc).

`double precision radialVelocityDispersion(\textcolor{red}{\textless type((treeNode))\textgreater}`
`node\arginout,\textcolor{red}{\textless double precision\textgreater} radius\argin)`
Returns the radial velocity dispersion (in km/s) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`double precision radialVelocityDispersionNumerical(<type((treeNode))> node↔,<double precision>`
`radius`
`argin)` Returns the radial velocity dispersion (in km/s) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`double precision radiusEnclosingDensity(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} density\argin)` Returns the radius (in Mpc) enclosing a given density threshold (in $M_\odot \mathrm{Mpc}^{-3}$) in the dark matter profile of `node`.

`double precision radiusEnclosingDensityNumerical(<type((treeNode))> node↔,<double precision> density→)` Returns the radius (in Mpc) in the dark matter profile of `node` which encloses the given `density` (given in units of $M_\odot/\mathrm{Mpc}^{-3}$).

`double precision radiusEnclosingMass(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} mass\argin)` Returns the radius (in Mpc) enclosing a given mass (in $M_\odot$) in the dark matter profile of `node`.

`double precision radiusEnclosingMassNumerical(<type((treeNode))> node↔,<double precision> mass→)` Returns the radius (in Mpc) in the dark matter profile of `node` which encloses the given `mass` (given in units of $M_\odot$).

`double precision radiusFromSpecificAngularMomentum(\textcolor{red}{\textless type((treeNode))\textgreat node\arginout,\textcolor{red}{\textless double precision\textgreater} specificAngularMomentum\argin` Returns the radius (in Mpc) in the dark matter profile of `node` at which the specific angular momentum of a circular orbit equals `specificAngularMomentum` (specified in units of km s^{-1} Mpc.

`double precision radiusFromSpecificAngularMomentumNumerical(<type((treeNode))> node↔,<double precision> specificAngularMomentum→)` Returns the radius (in Mpc) in `node` at which a circular orbit has the given `specificAngularMomentum` (given in units of km s^{-1} Mpc).

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`double precision rotationNormalization(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Returns the relation between specific angular momentum and rotation velocity (assuming a rotation velocity that is constant in radius) for the given `node`. Specifically, the normalization, A, returned is such that $V_\mathrm{rot} = AJ/M$

`double precision rotationNormalizationNumerical(<type((treeNode))> node↔)` Return the normalization of the rotation velocity vs. specific angular momentum relation.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

darkMatterProfileDMOEinasto

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`double precision circularVelocity(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} radius\argin)` Returns the circular velocity (in km/s) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`double precision circularVelocityMaximum(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Returns the maximum circular velocity (in km/s) in the dark matter profile of `node`.

`double precision circularVelocityMaximumNumerical(<type((treeNode))> node↔)` Returns the maximum circular velocity (in km/s) in the dark matter profile of `node`.

`double precision circularVelocityNumerical(<type((treeNode))> node↔,<double precision> radius argin)` Returns the circular velocity (in km/s) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`void deepCopy(\textcolor{red}{\textless class((darkMatterProfileDMOClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`double precision density(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcol double precision\textgreater} radius\argin)` Returns the density (in $M_\odot$ Mpc^{-3}) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`double precision densityLogSlope(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} radius\argin)` Returns the logarithmic slope of the density profile in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`double precision densityLogSlopeNumerical(<type((treeNode))> node↔,<double precision> radius→)` Returns the logarithmic slope of the density in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`<double> densityScaleFree(<double> radius→, <double> concentration→, <double> alpha→)` Returns the density (in units such that the virial mass and scale length are unity) in an Einasto dark matter profile with given `concentration` and `alpha` at the given `radius` (given in units of the scale radius).

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision enclosedMass(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\te double precision\textgreater} radius\argin)` Returns the enclosed mass (in $M_\odot$) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`double precision enclosedMassDifferenceNumerical(<type((treeNode))> node↔, <double precision> radiusLower→, <double precision> radiusUpper→)` Returns the enclosed mass difference (in $M_\odot$) in the dark matter profile of `node` between the given `radiusLower` and `radiusUpper` (given in units of Mpc) using a numerical calculation.

double precision enclosedMassNumerical(<type((treeNode))> node↔,<double precision> radius→) Returns the enclosed mass (in $M_\odot$) in the dark matter profile of node at the given radius (given in units of Mpc) using a numerical calculation.

<double> enclosedMassScaleFree(<double> radius→, <double> concentration→, <double> alpha→) Returns the enclosed mass (in units of the virial mass) in an Einasto dark matter profile with given concentration at the given radius (given in units of the scale radius).

double precision energy(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Return the total energy for the given node in units of $M_\odot$ km^2 s^{-2}.

double precision energyGrowthRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the rate of chance of the total energy of node in units of $M_\odot$ km^2 s^{-2} Gyr^{-1}.

double precision energyGrowthRateNumerical(<type((treeNode))> node↔) Return the rate of growth of the energy of the dark matter density profile.

double precision energyNumerical(<type((treeNode))> node↔) Return the energy of the dark matter density profile.

<void> energyTableMake(<double> concentrationRequired→, <double> alphaRequired→) Create a tabulation of the energy of Einasto profiles as a function of their concentration of α parameter.

<void> fourierProfileTableMake(<double> wavenumberRequired→, <double> concentrationRequired→, <double> alphaRequired→) Create a tabulation of the Fourier transform of Einasto profiles as a function of their α parameter and dimensionless wavenumber.

double precision freefallRadius(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} time\argin) Returns the freefall radius (in Mpc) corresponding to the given time (in Gyr) in node.

double precision freeFallRadiusIncreaseRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} time\argin) Returns the rate of increase of the freefall radius (in Mpc/Gyr) corresponding to the given time (in Gyr) in node.

double precision freefallRadiusIncreaseRateNumerical(<type((treeNode))> node↔,<double precision> time→) Returns the rate of increase of the freefall radius in the dark matter density profile at the specified time (given in Gyr).

double precision freefallRadiusNumerical(<type((treeNode))> node↔,<double precision> time→) Returns the freefall radius in the dark matter density profile at the specified time (given in Gyr).

<void> freefallTabulate(<double> freefallTimeScaleFree→, <double> alphaRequired→) Tabulates the freefall time vs. freefall radius for Einasto halos.

<double> freefallTimeScaleFree(<double> radius→, <double> alpha→) Compute the freefall time in a scale-free Einasto halo.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

`double precision kSpace(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolo double precision\textgreater} wavenumber\argin)` Returns the normalized Fourier space density profile of the dark matter profile of `node` at the given `waveNumber` (given in units of Mpc^{-1}).

`double precision kSpaceNumerical(<type((treeNode))> node↔, <double> waveNumber→)` Returns the Fourier transform of the adiabaticGnedin2004 density profile at the specified `waveNumber` (given in Mpc^{-1}).

`type(varying_string) objectType()` Return the type of the object.

`double precision potential(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textc double precision\textgreater} radius\argin,\textcolor{red}{\textless integer\textgreater} status\argout)` Returns the gravitational potential (in $(\mathrm{km/s})^2$) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`double precision potentialDifferenceNumerical(<type((treeNode))> node↔, <double precision> radiusLower→, <double precision> radiusUpper→)` Returns the gravitational potential difference (in $(\mathrm{km/s})^2$) in the dark matter profile of `node` between the given `radiusLower` and `radiusUpper` (given in units of Mpc) using a numerical calculation.

`double precision potentialNumerical(<type((treeNode))> node↔,<double precision> radius→,<integer> status←)` Returns the gravitational potential (in $(\mathrm{km/s})^2$) in the dark matter profile of `node` at the given `radius` (given in units of Mpc) using a numerical calculation.

`<double> potentialScaleFree(<double> radius→, <double> concentration→, <double> alpha→)` Returns the gravitational potential (in units where the virial mass and scale radius are unity) in an Einasto dark matter profile with given `concentration` and `alpha` at the given `radius` (given in units of the scale radius).

`double precision radialMoment(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\te double precision\textgreater} moment\argin,\textcolor{red}{\textless double precision\textgreater} radiusMinimum\argin,\textcolor{red}{\textless double precision\textgreater} radiusMaximum\argin)` Returns the $\mathtt{m}^{\mathrm{th}}$ radial moment of the dark matter profile of `node` optionally between the given `radiusMinimum` and `radiusMaximum` (given in units of Mpc).

`double precision radialMomentNumerical(<type((treeNode))> node↔, <double> moment→, <double> radiusMinimum→, <double> radiusMaximum→)` Returns the radial moment of the density in the dark matter profile of `node` between the given `radiusMinimum` and `radiusMaximum` (given in units of Mpc).

`double precision radialVelocityDispersion(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} radius\argin)` Returns the radial velocity dispersion (in km/s) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`double precision radialVelocityDispersionNumerical(<type((treeNode))> node↔,<double precision> radius argin)` Returns the radial velocity dispersion (in km/s) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`<double> radialVelocityDispersionScaleFree(<double> radius→, <double> alpha→)` Returns the scale-free radial velocity dispersion in an Einasto dark matter profile.

<void> radialVelocityDispersionTabulate(<double> radius→, <double> alphaRequired→) Tabulates the radial velocity dispersion vs. radius for Einasto halos.

double precision radiusEnclosingDensity(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} density\argin) Returns the radius (in Mpc) enclosing a given density threshold (in $M_\odot \mathrm{Mpc}^{-3}$) in the dark matter profile of node.

double precision radiusEnclosingDensityNumerical(<type((treeNode))> node↔,<double precision> density→) Returns the radius (in Mpc) in the dark matter profile of node which encloses the given density (given in units of $M_\odot/\mathrm{Mpc}^{-3}$).

double precision radiusEnclosingMass(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} mass\argin) Returns the radius (in Mpc) enclosing a given mass (in $M_\odot$) in the dark matter profile of node.

double precision radiusEnclosingMassNumerical(<type((treeNode))> node↔,<double precision> mass→) Returns the radius (in Mpc) in the dark matter profile of node which encloses the given mass (given in units of $M_\odot$).

double precision radiusFromSpecificAngularMomentum(\textcolor{red}{\textless type((treeNode))\textgreat node\arginout,\textcolor{red}{\textless double precision\textgreater} specificAngularMomentum\argin Returns the radius (in Mpc) in the dark matter profile of node at which the specific angular momentum of a circular orbit equals specificAngularMomentum (specified in units of km s^{-1} Mpc.

double precision radiusFromSpecificAngularMomentumNumerical(<type((treeNode))> node↔,<double precision> specificAngularMomentum→) Returns the radius (in Mpc) in node at which a circular orbit has the given specificAngularMomentum (given in units of km s^{-1} Mpc).

<double> radiusFromSpecificAngularMomentumScaleFree(<double> alpha→, <double> specificAngularMomentumS Comptue the radius at which a circular orbit has the given specificAngularMomentumScaleFree in a scale free Einasto profile.

<void> radiusFromSpecificAngularMomentumTableMake(<double> alphaRequired→, <double> specificAngularMom Create a tabulation of the relation between specific angular momentum and radius in an Einasto profile.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

double precision rotationNormalization(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the relation between specific angular momentum and rotation velocity (assuming a rotation velocity that is constant in radius) for the given node. Specifically, the normalization, A, returned is such that $V_{\rm rot} = AJ/M$

double precision rotationNormalizationNumerical(<type((treeNode))> node↔) Return the normalization of the rotation velocity vs. specific angular momentum relation.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

darkMatterProfileDMOHeated

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

<void> calculationReset(<type(table)> node↔) Reset memoized calculations.

double precision circularVelocity(\textcolor{red}{\textless type((treeNode))\textgreater}
node\arginout,\textcolor{red}{\textless double precision\textgreater} radius\argin)
Returns the circular velocity (in km/s) in the dark matter profile of node at the given radius (given in units of Mpc).

double precision circularVelocityMaximum(\textcolor{red}{\textless type((treeNode))\textgreater}
node\arginout) Returns the maximum circular velocity (in km/s) in the dark matter profile of node.

double precision circularVelocityMaximumNumerical(<type((treeNode))> node↔) Returns the maximum circular velocity (in km/s) in the dark matter profile of node.

double precision circularVelocityNumerical(<type((treeNode))> node↔,<double precision>
radius
argin) Returns the circular velocity (in km/s) in the dark matter profile of node at the given radius (given in units of Mpc).

void deepCopy(\textcolor{red}{\textless class((darkMatterProfileDMOClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

double precision density(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcol
double precision\textgreater} radius\argin) Returns the density (in $M_\odot$ Mpc^{-3}) in the dark matter profile of node at the given radius (given in units of Mpc).

double precision densityLogSlope(\textcolor{red}{\textless type((treeNode))\textgreater}
node\arginout,\textcolor{red}{\textless double precision\textgreater} radius\argin)
Returns the logarithmic slope of the density profile in the dark matter profile of node at the given radius (given in units of Mpc).

double precision densityLogSlopeNumerical(<type((treeNode))> node↔,<double precision>
radius→) Returns the logarithmic slope of the density in the dark matter profile of node at the given radius (given in units of Mpc).

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision enclosedMass(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\te
double precision\textgreater} radius\argin) Returns the enclosed mass (in $M_\odot$) in the dark matter profile of node at the given radius (given in units of Mpc).

`double precision enclosedMassDifferenceNumerical(<type((treeNode))> node↔, <double precision> radiusLower→, <double precision> radiusUpper→)` Returns the enclosed mass difference (in $M_\odot$) in the dark matter profile of `node` between the given `radiusLower` and `radiusUpper` (given in units of Mpc) using a numerical calculation.

`double precision enclosedMassNumerical(<type((treeNode))> node↔,<double precision> radius→)` Returns the enclosed mass (in $M_\odot$) in the dark matter profile of `node` at the given `radius` (given in units of Mpc) using a numerical calculation.

`double precision energy(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Return the total energy for the given `node` in units of $M_\odot$ km^2 s^{-2}.

`double precision energyGrowthRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Returns the rate of chance of the total energy of `node` in units of $M_\odot$ km^2 s^{-2} Gyr^{-1}.

`double precision energyGrowthRateNumerical(<type((treeNode))> node↔)` Return the rate of growth of the energy of the dark matter density profile.

`double precision energyNumerical(<type((treeNode))> node↔)` Return the energy of the dark matter density profile.

`double precision freefallRadius(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} time\argin)` Returns the freefall radius (in Mpc) corresponding to the given `time` (in Gyr) in `node`.

`double precision freeFallRadiusIncreaseRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} time\argin)` Returns the rate of increase of the freefall radius (in Mpc/Gyr) corresponding to the given `time` (in Gyr) in `node`.

`double precision freefallRadiusIncreaseRateNumerical(<type((treeNode))> node↔,<double precision> time→)` Returns the rate of increase of the freefall radius in the dark matter density profile at the specified `time` (given in Gyr).

`double precision freefallRadiusNumerical(<type((treeNode))> node↔,<double precision> time→)` Returns the freefall radius in the dark matter density profile at the specified `time` (given in Gyr).

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision kSpace(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolo double precision\textgreater} wavenumber\argin)` Returns the normalized Fourier space density profile of the dark matter profile of `node` at the given `waveNumber` (given in units of Mpc^{-1}).

`double precision kSpaceNumerical(<type((treeNode))> node↔, <double> waveNumber→)` Returns the Fourier transform of the adiabaticGnedin2004 density profile at the specified `waveNumber` (given in Mpc^{-1}).

`type(varying_string) objectType()` Return the type of the object.

`double precision potential(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\texto double precision\textgreater} radius\argin,\textcolor{red}{\textless integer\textgreater} status\argout)` Returns the gravitational potential (in $(\mathrm{km/s})^2$) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`double precision potentialDifferenceNumerical(<type((treeNode))> node↔, <double precision> radiusLower→, <double precision> radiusUpper→)` Returns the gravitational potential difference (in $(\mathrm{km/s})^2$) in the dark matter profile of `node` between the given `radiusLower` and `radiusUpper` (given in units of Mpc) using a numerical calculation.

`double precision potentialNumerical(<type((treeNode))> node↔,<double precision> radius→,<integer> status←)` Returns the gravitational potential (in $(\mathrm{km/s})^2$) in the dark matter profile of `node` at the given `radius` (given in units of Mpc) using a numerical calculation.

`double precision radialMoment(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\te double precision\textgreater} moment\argin,\textcolor{red}{\textless double precision\textgreater} radiusMinimum\argin,\textcolor{red}{\textless double precision\textgreater} radiusMaximum\argin)` Returns the $\mathtt{m}^{\mathrm{th}}$ radial moment of the dark matter profile of `node` optionally between the given `radiusMinimum` and `radiusMaximum` (given in units of Mpc).

`double precision radialMomentNumerical(<type((treeNode))> node↔, <double> moment→, <double> radiusMinimum→, <double> radiusMaximum→)` Returns the radial moment of the density in the dark matter profile of `node` between the given `radiusMinimum` and `radiusMaximum` (given in units of Mpc).

`double precision radialVelocityDispersion(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} radius\argin)` Returns the radial velocity dispersion (in km/s) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`double precision radialVelocityDispersionNumerical(<type((treeNode))> node↔,<double precision> radius argin)` Returns the radial velocity dispersion (in km/s) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`double precision radiusEnclosingDensity(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} density\argin)` Returns the radius (in Mpc) enclosing a given density threshold (in $M_\odot\mathrm{Mpc}^{-3}$) in the dark matter profile of `node`.

`double precision radiusEnclosingDensityNumerical(<type((treeNode))> node↔,<double precision> density→)` Returns the radius (in Mpc) in the dark matter profile of `node` which encloses the given `density` (given in units of $M_\odot/\mathrm{Mpc}^{-3}$).

`double precision radiusEnclosingMass(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} mass\argin)` Returns the radius (in Mpc) enclosing a given mass (in $M_\odot$) in the dark matter profile of `node`.

`double precision radiusEnclosingMassNumerical(<type((treeNode))> node↔,<double precision> mass→)` Returns the radius (in Mpc) in the dark matter profile of `node` which encloses the given `mass` (given in units of $M_\odot$).

`double precision radiusFromSpecificAngularMomentum(\textcolor{red}{\textless type((treeNode))\textgreat node\arginout,\textcolor{red}{\textless double precision\textgreater} specificAngularMomentum\argin` Returns the radius (in Mpc) in the dark matter profile of `node` at which the specific angular momentum of a circular orbit equals `specificAngularMomentum` (specified in units of km s^{-1} Mpc.

`double precision radiusFromSpecificAngularMomentumNumerical(<type((treeNode))> node↔,<double precision> specificAngularMomentum→)` Returns the radius (in Mpc) in `node` at which a circular orbit has the given `specificAngularMomentum` (given in units of km s^{-1} Mpc).

`<double> radiusInitial(<*type(treeNode)> node↔, <double> radiusFinal→)` Return the initial radius corresponding to the given final radius in a heated dark matter halo density profile.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`double precision rotationNormalization(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Returns the relation between specific angular momentum and rotation velocity (assuming a rotation velocity that is constant in radius) for the given `node`. Specifically, the normalization, A, returned is such that $V_{\rm rot} = AJ/M$

`double precision rotationNormalizationNumerical(<type((treeNode))> node↔)` Return the normalization of the rotation velocity vs. specific angular momentum relation.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

darkMatterProfileDMOIsothermal

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`double precision circularVelocity(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} radius\argin)` Returns the circular velocity (in km/s) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`double precision circularVelocityMaximum(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Returns the maximum circular velocity (in km/s) in the dark matter profile of `node`.

`double precision circularVelocityMaximumNumerical(<type((treeNode))> node↔)` Returns the maximum circular velocity (in km/s) in the dark matter profile of `node`.

`double precision circularVelocityNumerical(<type((treeNode))> node↔,<double precision> radius argin)` Returns the circular velocity (in km/s) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`void deepCopy(\textcolor{red}{\textless class((darkMatterProfileDMOClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`double precision density(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcol double precision\textgreater} radius\argin)` Returns the density (in $M_\odot$ Mpc^{-3}) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`double precision densityLogSlope(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} radius\argin)` Returns the logarithmic slope of the density profile in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`double precision densityLogSlopeNumerical(<type((treeNode))> node↔,<double precision> radius→)` Returns the logarithmic slope of the density in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision enclosedMass(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\te double precision\textgreater} radius\argin)` Returns the enclosed mass (in $M_\odot$) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`double precision enclosedMassDifferenceNumerical(<type((treeNode))> node↔, <double precision> radiusLower→, <double precision> radiusUpper→)` Returns the enclosed mass difference (in $M_\odot$) in the dark matter profile of `node` between the given `radiusLower` and `radiusUpper` (given in units of Mpc) using a numerical calculation.

`double precision enclosedMassNumerical(<type((treeNode))> node↔,<double precision> radius→)` Returns the enclosed mass (in $M_\odot$) in the dark matter profile of `node` at the given `radius` (given in units of Mpc) using a numerical calculation.

`double precision energy(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Return the total energy for the given `node` in units of $M_\odot$ km^2 s^{-2}.

`double precision energyGrowthRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Returns the rate of chance of the total energy of `node` in units of $M_\odot$ km^2 s^{-2} Gyr^{-1}.

`double precision energyGrowthRateNumerical(<type((treeNode))> node↔)` Return the rate of growth of the energy of the dark matter density profile.

`double precision energyNumerical(<type((treeNode))> node↔)` Return the energy of the dark matter density profile.

`double precision freefallRadius(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} time\argin)` Returns the freefall radius (in Mpc) corresponding to the given `time` (in Gyr) in `node`.

`double precision freeFallRadiusIncreaseRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} time\argin)` Returns the rate of increase of the freefall radius (in Mpc/Gyr) corresponding to the given `time` (in Gyr) in `node`.

`double precision freefallRadiusIncreaseRateNumerical(<type((treeNode))> node↔,<double precision> time→)` Returns the rate of increase of the freefall radius in the dark matter density profile at the specified `time` (given in Gyr).

`double precision freefallRadiusNumerical(<type((treeNode))> node↔,<double precision> time→)` Returns the freefall radius in the dark matter density profile at the specified `time` (given in Gyr).

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision kSpace(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolo double precision\textgreater} wavenumber\argin)` Returns the normalized Fourier space density profile of the dark matter profile of `node` at the given `waveNumber` (given in units of Mpc^{-1}).

`double precision kSpaceNumerical(<type((treeNode))> node↔, <double> waveNumber→)` Returns the Fourier transform of the adiabaticGnedin2004 density profile at the specified `waveNumber` (given in Mpc^{-1}).

`type(varying_string) objectType()` Return the type of the object.

`double precision potential(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textc double precision\textgreater} radius\argin,\textcolor{red}{\textless integer\textgreater} status\argout)` Returns the gravitational potential (in $(\mathrm{km/s})^2$) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`double precision potentialDifferenceNumerical(<type((treeNode))> node↔, <double precision> radiusLower→, <double precision> radiusUpper→)` Returns the gravitational potential difference (in $(\mathrm{km/s})^2$) in the dark matter profile of `node` between the given `radiusLower` and `radiusUpper` (given in units of Mpc) using a numerical calculation.

`double precision potentialNumerical(<type((treeNode))> node↔,<double precision> radius→,<integer> status←)` Returns the gravitational potential (in $(\mathrm{km/s})^2$) in the dark matter profile of `node` at the given `radius` (given in units of Mpc) using a numerical calculation.

`double precision radialMoment(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\te double precision\textgreater} moment\argin,\textcolor{red}{\textless double precision\textgreater} radiusMinimum\argin,\textcolor{red}{\textless double precision\textgreater} radiusMaximum\argin)` Returns the $\mathtt{m}^{\mathrm{th}}$ radial moment of the dark matter profile of `node` optionally between the given `radiusMinimum` and `radiusMaximum` (given in units of Mpc).

`double precision radialMomentNumerical(<type((treeNode))> node↔, <double> moment→, <double> radiusMinimum→, <double> radiusMaximum→)` Returns the radial moment of the density in the dark matter profile of `node` between the given `radiusMinimum` and `radiusMaximum` (given in units of Mpc).

`double precision radialVelocityDispersion(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} radius\argin)` Returns the radial velocity dispersion (in km/s) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`double precision radialVelocityDispersionNumerical(<type((treeNode))> node↔,<double precision> radius argin)` Returns the radial velocity dispersion (in km/s) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`double precision radiusEnclosingDensity(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} density\argin)` Returns the radius (in Mpc) enclosing a given density threshold (in $M_\odot \mathrm{Mpc}^{-3}$) in the dark matter profile of `node`.

`double precision radiusEnclosingDensityNumerical(<type((treeNode))> node↔,<double precision> density→)` Returns the radius (in Mpc) in the dark matter profile of `node` which encloses the given `density` (given in units of $M_\odot/\mathrm{Mpc}^{-3}$).

`double precision radiusEnclosingMass(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} mass\argin)` Returns the radius (in Mpc) enclosing a given mass (in $M_\odot$) in the dark matter profile of `node`.

`double precision radiusEnclosingMassNumerical(<type((treeNode))> node↔,<double precision> mass→)` Returns the radius (in Mpc) in the dark matter profile of `node` which encloses the given `mass` (given in units of $M_\odot$).

`double precision radiusFromSpecificAngularMomentum(\textcolor{red}{\textless type((treeNode))\textgreat node\arginout,\textcolor{red}{\textless double precision\textgreater} specificAngularMomentum\argin` Returns the radius (in Mpc) in the dark matter profile of `node` at which the specific angular momentum of a circular orbit equals `specificAngularMomentum` (specified in units of km s^{-1} Mpc.

`double precision radiusFromSpecificAngularMomentumNumerical(<type((treeNode))> node↔,<double precision> specificAngularMomentum→)` Returns the radius (in Mpc) in `node` at which a circular orbit has the given `specificAngularMomentum` (given in units of km s^{-1} Mpc).

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`double precision rotationNormalization(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Returns the relation between specific angular momentum and rotation velocity (assuming a rotation velocity that is constant in radius) for the given `node`. Specifically, the normalization, A, returned is such that $V_{\mathrm{rot}} = AJ/M$

`double precision rotationNormalizationNumerical(<type((treeNode))> node↔)` Return the normalization of the rotation velocity vs. specific angular momentum relation.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

darkMatterProfileDMONFW

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

<double> angularMomentumScaleFree(<double> concentration→) Returns the total angular momentum in an NFW dark matter profile with given concentration.

void autoHook() Insert any event hooks required by this object.

<void> calculationReset(<type(table)> node↔) Reset memoized calculations.

double precision circularVelocity(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} radius\argin) Returns the circular velocity (in km/s) in the dark matter profile of node at the given radius (given in units of Mpc).

double precision circularVelocityMaximum(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the maximum circular velocity (in km/s) in the dark matter profile of node.

double precision circularVelocityMaximumNumerical(<type((treeNode))> node↔) Returns the maximum circular velocity (in km/s) in the dark matter profile of node.

double precision circularVelocityNumerical(<type((treeNode))> node↔,<double precision> radius
argin) Returns the circular velocity (in km/s) in the dark matter profile of node at the given radius (given in units of Mpc).

void deepCopy(\textcolor{red}{\textless class((darkMatterProfileDMOClass))\textgreater} destination\arginout) Perform a deep copy of the object.

double precision density(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcol double precision\textgreater} radius\argin) Returns the density (in $M_\odot$ Mpc^{-3}) in the dark matter profile of node at the given radius (given in units of Mpc).

<double> densityEnclosedByRadiusScaleFree(<double> radius→, <double> concentration→) Returns the density (in units of the virial mass per cubic scale radius) in an NFW dark matter profile with given concentration which is enclosed a given radius (in units of the scale radius).

double precision densityLogSlope(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} radius\argin) Returns the logarithmic slope of the density profile in the dark matter profile of node at the given radius (given in units of Mpc).

double precision densityLogSlopeNumerical(<type((treeNode))> node↔,<double precision> radius→) Returns the logarithmic slope of the density in the dark matter profile of node at the given radius (given in units of Mpc).

<double> densityScaleFree(<double> radius→, <double> concentration→) Returns the density (in units such that the virial mass and scale length are unity) in an NFW dark matter profile with given concentration and alpha at the given radius (given in units of the scale radius).

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

<void> enclosedDensityTabulate(<double> enclosedDensityScaleFree→) Tabulate the density enclosed within a given radius for the NFW profile.

double precision enclosedMass(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\te double precision\textgreater} radius\argin) Returns the enclosed mass (in $M_\odot$) in the dark matter profile of node at the given radius (given in units of Mpc).

double precision enclosedMassDifferenceNumerical(<type((treeNode))> node↔, <double precision> radiusLower→, <double precision> radiusUpper→) Returns the enclosed mass difference (in $M_\odot$) in the dark matter profile of node between the given radiusLower and radiusUpper (given in units of Mpc) using a numerical calculation.

double precision enclosedMassNumerical(<type((treeNode))> node↔,<double precision> radius→) Returns the enclosed mass (in $M_\odot$) in the dark matter profile of node at the given radius (given in units of Mpc) using a numerical calculation.

<double> enclosedMassScaleFree(<double> radius→, <double> concentration→) Returns the enclosed mass (in units of the virial mass) in an NFW dark matter profile with given concentration at the given radius (given in units of the scale radius).

double precision energy(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Return the total energy for the given node in units of $M_\odot$ km^2 s^{-2}.

double precision energyGrowthRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the rate of chance of the total energy of node in units of $M_\odot$ km^2 s^{-2} Gyr^{-1}.

double precision energyGrowthRateNumerical(<type((treeNode))> node↔) Return the rate of growth of the energy of the dark matter density profile.

double precision energyNumerical(<type((treeNode))> node↔) Return the energy of the dark matter density profile.

double precision freefallRadius(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} time\argin) Returns the freefall radius (in Mpc) corresponding to the given time (in Gyr) in node.

double precision freeFallRadiusIncreaseRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} time\argin) Returns the rate of increase of the freefall radius (in Mpc/Gyr) corresponding to the given time (in Gyr) in node.

double precision freefallRadiusIncreaseRateNumerical(<type((treeNode))> node↔,<double precision> time→) Returns the rate of increase of the freefall radius in the dark matter density profile at the specified time (given in Gyr).

double precision freefallRadiusNumerical(<type((treeNode))> node↔,<double precision> time→) Returns the freefall radius in the dark matter density profile at the specified time (given in Gyr).

<void> freefallTabulate(<double> freefallTimeScaleFree→) Tabulates the freefall time vs. freefall radius for NFW halos.

<double> freefallTimeScaleFree(<double> radius→) Compute the freefall time in a scale-free NFW halo.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<void> inverseAngularMomentum(<double> specificAngularMomentum→) Tabulates the specific angular momentum vs. radius in an NFW profile for rapid inversion.

<logical> isDefault() Return true if this is the default object of this class.

double precision kSpace(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolo double precision\textgreater} wavenumber\argin) Returns the normalized Fourier space density profile of the dark matter profile of node at the given waveNumber (given in units of Mpc^{-1}).

double precision kSpaceNumerical(<type((treeNode))> node↔, <double> waveNumber→) Returns the Fourier transform of the adiabaticGnedin2004 density profile at the specified waveNumber (given in Mpc^{-1}).

type(varying_string) objectType() Return the type of the object.

double precision potential(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\texto double precision\textgreater} radius\argin,\textcolor{red}{\textless integer\textgreater} status\argout) Returns the gravitational potential (in $(\mathrm{km/s})^2$) in the dark matter profile of node at the given radius (given in units of Mpc).

double precision potentialDifferenceNumerical(<type((treeNode))> node↔, <double precision> radiusLower→, <double precision> radiusUpper→) Returns the gravitational potential difference (in $(\mathrm{km/s})^2$) in the dark matter profile of node between the given radiusLower and radiusUpper (given in units of Mpc) using a numerical calculation.

double precision potentialNumerical(<type((treeNode))> node↔,<double precision> radius→,<integer> status←) Returns the gravitational potential (in $(\mathrm{km/s})^2$) in the dark matter profile of node at the given radius (given in units of Mpc) using a numerical calculation.

<double> profileEnergy(<double> concentration→) Computes the total energy of an NFW profile halo of given concentration.

double precision radialMoment(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\te double precision\textgreater} moment\argin,\textcolor{red}{\textless double precision\textgreater} radiusMinimum\argin,\textcolor{red}{\textless double precision\textgreater} radiusMaximum\argin) Returns the $\mathtt{m}^{\mathrm{th}}$ radial moment of the dark matter profile of node optionally between the given radiusMinimum and radiusMaximum (given in units of Mpc).

double precision radialMomentNumerical(<type((treeNode))> node↔, <double> moment→, <double> radiusMinimum→, <double> radiusMaximum→) Returns the radial moment of the density in the dark matter profile of node between the given radiusMinimum and radiusMaximum (given in units of Mpc).

`double precision radialVelocityDispersion(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} radius\argin)` Returns the radial velocity dispersion (in km/s) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`double precision radialVelocityDispersionNumerical(<type((treeNode))> node↔,<double precision> radius argin)` Returns the radial velocity dispersion (in km/s) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`<double> radialVelocityDispersionScaleFree(<double> radius→, <double> concentration→)` Returns the radial velocity dispersion (in units of the virial velocity) in an NFW dark matter profile with given `concentration` at the given `radius` (given in units of the scale radius).

`double precision radiusEnclosingDensity(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} density\argin)` Returns the radius (in Mpc) enclosing a given density threshold (in $M_\odot \mathrm{Mpc}^{-3}$) in the dark matter profile of `node`.

`double precision radiusEnclosingDensityNumerical(<type((treeNode))> node↔,<double precision> density→)` Returns the radius (in Mpc) in the dark matter profile of `node` which encloses the given `density` (given in units of $M_\odot/\mathrm{Mpc}^{-3}$).

`double precision radiusEnclosingMass(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} mass\argin)` Returns the radius (in Mpc) enclosing a given mass (in $M_\odot$) in the dark matter profile of `node`.

`double precision radiusEnclosingMassNumerical(<type((treeNode))> node↔,<double precision> mass→)` Returns the radius (in Mpc) in the dark matter profile of `node` which encloses the given `mass` (given in units of $M_\odot$).

`double precision radiusFromSpecificAngularMomentum(\textcolor{red}{\textless type((treeNode))\textgreat node\arginout,\textcolor{red}{\textless double precision\textgreater} specificAngularMomentum\argin` Returns the radius (in Mpc) in the dark matter profile of `node` at which the specific angular momentum of a circular orbit equals `specificAngularMomentum` (specified in units of km s^{-1} Mpc.

`double precision radiusFromSpecificAngularMomentumNumerical(<type((treeNode))> node↔,<double precision> specificAngularMomentum→)` Returns the radius (in Mpc) in `node` at which a circular orbit has the given `specificAngularMomentum` (given in units of km s^{-1} Mpc).

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`double precision rotationNormalization(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Returns the relation between specific angular momentum and rotation velocity (assuming a rotation velocity that is constant in radius) for the given `node`. Specifically, the normalization, A, returned is such that $V_{\rm rot} = AJ/M$

`double precision rotationNormalizationNumerical(<type((treeNode))> node↔)` Return the normalization of the rotation velocity vs. specific angular momentum relation.

<double> specificAngularMomentumScaleFree(<double> radius→) Returns the specific angular momentum, normalized to unit scale length and unit velocity at the scale radius, at position radius (in units of the scale radius) in an NFW profile.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

<void> tabulate(<double> concentration→) Tabulate properties of the NFW halo profile which must be computed numerically.

darkMatterProfileDMOTruncated

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

<void> calculationReset(<type(table)> node↔) Reset memoized calculations.

double precision circularVelocity(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} radius\argin) Returns the circular velocity (in km/s) in the dark matter profile of node at the given radius (given in units of Mpc).

double precision circularVelocityMaximum(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the maximum circular velocity (in km/s) in the dark matter profile of node.

double precision circularVelocityMaximumNumerical(<type((treeNode))> node↔) Returns the maximum circular velocity (in km/s) in the dark matter profile of node.

double precision circularVelocityNumerical(<type((treeNode))> node↔,<double precision> radius
argin) Returns the circular velocity (in km/s) in the dark matter profile of node at the given radius (given in units of Mpc).

void deepCopy(\textcolor{red}{\textless class((darkMatterProfileDMOClass))\textgreater} destination\arginout) Perform a deep copy of the object.

double precision density(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcol double precision\textgreater} radius\argin) Returns the density (in $M_\odot$ Mpc^{-3}) in the dark matter profile of node at the given radius (given in units of Mpc).

double precision densityLogSlope(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} radius\argin) Returns the logarithmic slope of the density profile in the dark matter profile of node at the given radius (given in units of Mpc).

`double precision densityLogSlopeNumerical(<type((treeNode))> node↔,<double precision> radius→)` Returns the logarithmic slope of the density in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision enclosedMass(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\te double precision\textgreater} radius\argin)` Returns the enclosed mass (in $M_\odot$) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`double precision enclosedMassDifferenceNumerical(<type((treeNode))> node↔, <double precision> radiusLower→, <double precision> radiusUpper→)` Returns the enclosed mass difference (in $M_\odot$) in the dark matter profile of `node` between the given `radiusLower` and `radiusUpper` (given in units of Mpc) using a numerical calculation.

`double precision enclosedMassNumerical(<type((treeNode))> node↔,<double precision> radius→)` Returns the enclosed mass (in $M_\odot$) in the dark matter profile of `node` at the given `radius` (given in units of Mpc) using a numerical calculation.

`double precision energy(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Return the total energy for the given `node` in units of $M_\odot$ km^2 s^{-2}.

`double precision energyGrowthRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Returns the rate of chance of the total energy of `node` in units of $M_\odot$ km^2 s^{-2} Gyr^{-1}.

`double precision energyGrowthRateNumerical(<type((treeNode))> node↔)` Return the rate of growth of the energy of the dark matter density profile.

`double precision energyNumerical(<type((treeNode))> node↔)` Return the energy of the dark matter density profile.

`double precision freefallRadius(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} time\argin)` Returns the freefall radius (in Mpc) corresponding to the given `time` (in Gyr) in `node`.

`double precision freeFallRadiusIncreaseRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} time\argin)` Returns the rate of increase of the freefall radius (in Mpc/Gyr) corresponding to the given `time` (in Gyr) in `node`.

`double precision freefallRadiusIncreaseRateNumerical(<type((treeNode))> node↔,<double precision> time→)` Returns the rate of increase of the freefall radius in the dark matter density profile at the specified `time` (given in Gyr).

`double precision freefallRadiusNumerical(<type((treeNode))> node↔,<double precision> time→)` Returns the freefall radius in the dark matter density profile at the specified `time` (given in Gyr).

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision kSpace(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolo double precision\textgreater} wavenumber\argin)` Returns the normalized Fourier space density profile of the dark matter profile of `node` at the given `waveNumber` (given in units of Mpc^{-1}).

`double precision kSpaceNumerical(<type((treeNode))> node↔, <double> waveNumber→)` Returns the Fourier transform of the adiabaticGnedin2004 density profile at the specified `waveNumber` (given in Mpc^{-1}).

`type(varying_string) objectType()` Return the type of the object.

`double precision potential(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textc double precision\textgreater} radius\argin,\textcolor{red}{\textless integer\textgreater} status\argout)` Returns the gravitational potential (in $(\mathrm{km/s})^2$) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`double precision potentialDifferenceNumerical(<type((treeNode))> node↔, <double precision> radiusLower→, <double precision> radiusUpper→)` Returns the gravitational potential difference (in $(\mathrm{km/s})^2$) in the dark matter profile of `node` between the given `radiusLower` and `radiusUpper` (given in units of Mpc) using a numerical calculation.

`double precision potentialNumerical(<type((treeNode))> node↔,<double precision> radius→,<integer> status←)` Returns the gravitational potential (in $(\mathrm{km/s})^2$) in the dark matter profile of `node` at the given `radius` (given in units of Mpc) using a numerical calculation.

`double precision radialMoment(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\te double precision\textgreater} moment\argin,\textcolor{red}{\textless double precision\textgreater} radiusMinimum\argin,\textcolor{red}{\textless double precision\textgreater} radiusMaximum\argin)` Returns the $\mathtt{m}^{\mathrm{th}}$ radial moment of the dark matter profile of `node` optionally between the given `radiusMinimum` and `radiusMaximum` (given in units of Mpc).

`double precision radialMomentNumerical(<type((treeNode))> node↔, <double> moment→, <double> radiusMinimum→, <double> radiusMaximum→)` Returns the radial moment of the density in the dark matter profile of `node` between the given `radiusMinimum` and `radiusMaximum` (given in units of Mpc).

`double precision radialVelocityDispersion(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} radius\argin)` Returns the radial velocity dispersion (in km/s) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`double precision radialVelocityDispersionNumerical(<type((treeNode))> node↔,<double precision> radius argin)` Returns the radial velocity dispersion (in km/s) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`double precision radiusEnclosingDensity(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} density\argin)` Returns the radius (in Mpc) enclosing a given density threshold (in $M_\odot\mathrm{Mpc}^{-3}$) in the dark matter profile of `node`.

`double precision radiusEnclosingDensityNumerical(<type((treeNode))> node↔,<double precision> density→)` Returns the radius (in Mpc) in the dark matter profile of `node` which encloses the given `density` (given in units of $M_\odot/\mathrm{Mpc}^{-3}$).

`double precision radiusEnclosingMass(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} mass\argin)` Returns the radius (in Mpc) enclosing a given mass (in $M_\odot$) in the dark matter profile of `node`.

`double precision radiusEnclosingMassNumerical(<type((treeNode))> node↔,<double precision> mass→)` Returns the radius (in Mpc) in the dark matter profile of `node` which encloses the given `mass` (given in units of $M_\odot$).

`double precision radiusFromSpecificAngularMomentum(\textcolor{red}{\textless type((treeNode))\textgreat node\arginout,\textcolor{red}{\textless double precision\textgreater} specificAngularMomentum\argin` Returns the radius (in Mpc) in the dark matter profile of `node` at which the specific angular momentum of a circular orbit equals `specificAngularMomentum` (specified in units of km s^{-1} Mpc.

`double precision radiusFromSpecificAngularMomentumNumerical(<type((treeNode))> node↔,<double precision> specificAngularMomentum→)` Returns the radius (in Mpc) in `node` at which a circular orbit has the given `specificAngularMomentum` (given in units of km s^{-1} Mpc).

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`double precision rotationNormalization(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Returns the relation between specific angular momentum and rotation velocity (assuming a rotation velocity that is constant in radius) for the given `node`. Specifically, the normalization, A, returned is such that $V_{\rm rot} = AJ/M$

`double precision rotationNormalizationNumerical(<type((treeNode))> node↔)` Return the normalization of the rotation velocity vs. specific angular momentum relation.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`void truncationFunction(<type(treeNode)> node↔, <double> radius→, <double> [x]←, <double> [multiplier]←, <double> [multiplierGradient]←)` Returns the enclosed mass (in $M_\odot$) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

darkMatterProfileDMOTruncatedExponential

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`<void> calculationReset(<type(table)> node↔)` Reset memoized calculations.

double precision circularVelocity(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} radius\argin) Returns the circular velocity (in km/s) in the dark matter profile of node at the given radius (given in units of Mpc).

double precision circularVelocityMaximum(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the maximum circular velocity (in km/s) in the dark matter profile of node.

double precision circularVelocityMaximumNumerical(<type((treeNode))> node↔) Returns the maximum circular velocity (in km/s) in the dark matter profile of node.

double precision circularVelocityNumerical(<type((treeNode))> node↔,<double precision> radius
argin) Returns the circular velocity (in km/s) in the dark matter profile of node at the given radius (given in units of Mpc).

void deepCopy(\textcolor{red}{\textless class((darkMatterProfileDMOClass))\textgreater} destination\arginout) Perform a deep copy of the object.

double precision density(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcol double precision\textgreater} radius\argin) Returns the density (in $M_\odot$ Mpc^{-3}) in the dark matter profile of node at the given radius (given in units of Mpc).

double precision densityLogSlope(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} radius\argin) Returns the logarithmic slope of the density profile in the dark matter profile of node at the given radius (given in units of Mpc).

double precision densityLogSlopeNumerical(<type((treeNode))> node↔,<double precision> radius→) Returns the logarithmic slope of the density in the dark matter profile of node at the given radius (given in units of Mpc).

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision enclosedMass(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\te double precision\textgreater} radius\argin) Returns the enclosed mass (in $M_\odot$) in the dark matter profile of node at the given radius (given in units of Mpc).

double precision enclosedMassDifferenceNumerical(<type((treeNode))> node↔, <double precision> radiusLower→, <double precision> radiusUpper→) Returns the enclosed mass difference (in $M_\odot$) in the dark matter profile of node between the given radiusLower and radiusUpper (given in units of Mpc) using a numerical calculation.

double precision enclosedMassNumerical(<type((treeNode))> node↔,<double precision> radius→) Returns the enclosed mass (in $M_\odot$) in the dark matter profile of node at the given radius (given in units of Mpc) using a numerical calculation.

double precision energy(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Return the total energy for the given node in units of $M_\odot$ km^2 s^{-2}.

`double precision energyGrowthRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Returns the rate of chance of the total energy of `node` in units of $M_\odot$ km^2 s^{-2} Gyr^{-1}.

`double precision energyGrowthRateNumerical(<type((treeNode))> node↔)` Return the rate of growth of the energy of the dark matter density profile.

`double precision energyNumerical(<type((treeNode))> node↔)` Return the energy of the dark matter density profile.

`double precision freefallRadius(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} time\argin)` Returns the freefall radius (in Mpc) corresponding to the given `time` (in Gyr) in `node`.

`double precision freeFallRadiusIncreaseRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} time\argin)` Returns the rate of increase of the freefall radius (in Mpc/Gyr) corresponding to the given `time` (in Gyr) in `node`.

`double precision freefallRadiusIncreaseRateNumerical(<type((treeNode))> node↔,<double precision> time→)` Returns the rate of increase of the freefall radius in the dark matter density profile at the specified `time` (given in Gyr).

`double precision freefallRadiusNumerical(<type((treeNode))> node↔,<double precision> time→)` Returns the freefall radius in the dark matter density profile at the specified `time` (given in Gyr).

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision kSpace(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolo double precision\textgreater} wavenumber\argin)` Returns the normalized Fourier space density profile of the dark matter profile of `node` at the given `waveNumber` (given in units of Mpc^{-1}).

`double precision kSpaceNumerical(<type((treeNode))> node↔, <double> waveNumber→)` Returns the Fourier transform of the adiabaticGnedin2004 density profile at the specified `waveNumber` (given in Mpc^{-1}).

`type(varying_string) objectType()` Return the type of the object.

`double precision potential(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textc double precision\textgreater} radius\argin,\textcolor{red}{\textless integer\textgreater} status\argout)` Returns the gravitational potential (in (km/s)^2) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`double precision potentialDifferenceNumerical(<type((treeNode))> node↔, <double precision> radiusLower→, <double precision> radiusUpper→)` Returns the gravitational potential difference (in (km/s)^2) in the dark matter profile of `node` between the given `radiusLower` and `radiusUpper` (given in units of Mpc) using a numerical calculation.

`double precision potentialNumerical(<type((treeNode))> node↔,<double precision> radius→,<integer> status←)` Returns the gravitational potential (in (km/s)^2) in the dark matter profile of `node` at the given `radius` (given in units of Mpc) using a numerical calculation.

`double precision radialMoment(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\te double precision\textgreater} moment\argin,\textcolor{red}{\textless double precision\textgreater} radiusMinimum\argin,\textcolor{red}{\textless double precision\textgreater} radiusMaximum\argin)` Returns the $\mathtt{m}^{\mathrm{th}}$ radial moment of the dark matter profile of `node` optionally between the given `radiusMinimum` and `radiusMaximum` (given in units of Mpc).

`double precision radialMomentNumerical(<type((treeNode))> node↔, <double> moment→, <double> radiusMinimum→, <double> radiusMaximum→)` Returns the radial moment of the density in the dark matter profile of `node` between the given `radiusMinimum` and `radiusMaximum` (given in units of Mpc).

`double precision radialVelocityDispersion(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} radius\argin)` Returns the radial velocity dispersion (in km/s) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`double precision radialVelocityDispersionNumerical(<type((treeNode))> node↔,<double precision> radius argin)` Returns the radial velocity dispersion (in km/s) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`double precision radiusEnclosingDensity(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} density\argin)` Returns the radius (in Mpc) enclosing a given density threshold (in $M_\odot\mathrm{Mpc}^{-3}$) in the dark matter profile of `node`.

`double precision radiusEnclosingDensityNumerical(<type((treeNode))> node↔,<double precision> density→)` Returns the radius (in Mpc) in the dark matter profile of `node` which encloses the given `density` (given in units of $M_\odot/\mathrm{Mpc}^{-3}$).

`double precision radiusEnclosingMass(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} mass\argin)` Returns the radius (in Mpc) enclosing a given mass (in $M_\odot$) in the dark matter profile of `node`.

`double precision radiusEnclosingMassNumerical(<type((treeNode))> node↔,<double precision> mass→)` Returns the radius (in Mpc) in the dark matter profile of `node` which encloses the given `mass` (given in units of $M_\odot$).

`double precision radiusFromSpecificAngularMomentum(\textcolor{red}{\textless type((treeNode))\textgreat node\arginout,\textcolor{red}{\textless double precision\textgreater} specificAngularMomentum\argin` Returns the radius (in Mpc) in the dark matter profile of `node` at which the specific angular momentum of a circular orbit equals `specificAngularMomentum` (specified in units of km s^{-1} Mpc.

`double precision radiusFromSpecificAngularMomentumNumerical(<type((treeNode))> node↔,<double precision> specificAngularMomentum→)` Returns the radius (in Mpc) in `node` at which a circular orbit has the given `specificAngularMomentum` (given in units of km s^{-1} Mpc).

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`double precision rotationNormalization(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Returns the relation between specific angular momentum and rotation velocity (assuming a rotation velocity that is constant in radius) for the given `node`. Specifically, the normalization, A, returned is such that $V_{\rm rot} = AJ/M$

`double precision rotationNormalizationNumerical(<type((treeNode))> node↔)` Return the normalization of the rotation velocity vs. specific angular momentum relation.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`void truncationFunction(<type(treeNode)> node↔, <double> radius→, <double> [x]←, <double> [multiplier]←, <double> [multiplierGradient]←)` Returns the enclosed mass (in $M_\odot$) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

darkMatterProfileGeneric

`double precision circularVelocityMaximumNumerical(<type((treeNode))> node↔)` Returns the maximum circular velocity (in km/s) in the dark matter profile of `node`.

`double precision circularVelocityNumerical(<type((treeNode))> node↔,<double precision> radius`
`argin)` Returns the circular velocity (in km/s) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`double precision density(<type((treeNode))> node↔,<double precision> radius→)` Returns the density (in $M_\odot$/Mpc^3) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`double precision densityLogSlopeNumerical(<type((treeNode))> node↔,<double precision> radius→)` Returns the logarithmic slope of the density in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`double precision enclosedMass(<type((treeNode))> node↔,<double precision> radius→)` Returns the enclosed mass (in $M_\odot$) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`double precision enclosedMassDifferenceNumerical(<type((treeNode))> node↔, <double precision> radiusLower→, <double precision> radiusUpper→)` Returns the enclosed mass difference (in $M_\odot$) in the dark matter profile of `node` between the given `radiusLower` and `radiusUpper` (given in units of Mpc) using a numerical calculation.

`double precision enclosedMassNumerical(<type((treeNode))> node↔,<double precision> radius→)` Returns the enclosed mass (in $M_\odot$) in the dark matter profile of `node` at the given `radius` (given in units of Mpc) using a numerical calculation.

`double precision energyGrowthRateNumerical(<type((treeNode))> node↔)` Return the rate of growth of the energy of the dark matter density profile.

`double precision energyNumerical(<type((treeNode))> node↔)` Return the energy of the dark matter density profile.

`double precision freefallRadiusIncreaseRateNumerical(<type((treeNode))> node↔,<double precision> time→)` Returns the rate of increase of the freefall radius in the dark matter density profile at the specified `time` (given in Gyr).

`double precision freefallRadiusNumerical(<type((treeNode))> node↔,<double precision> time→)` Returns the freefall radius in the dark matter density profile at the specified `time` (given in Gyr).

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision kSpaceNumerical(<type((treeNode))> node↔, <double> waveNumber→)` Returns the Fourier transform of the adiabaticGnedin2004 density profile at the specified `waveNumber` (given in Mpc^{-1}).

`double precision potentialDifferenceNumerical(<type((treeNode))> node↔, <double precision> radiusLower→, <double precision> radiusUpper→)` Returns the gravitational potential difference (in $(\mathrm{km/s})^2$) in the dark matter profile of `node` between the given `radiusLower` and `radiusUpper` (given in units of Mpc) using a numerical calculation.

`double precision potentialNumerical(<type((treeNode))> node↔,<double precision> radius→,<integer> status←)` Returns the gravitational potential (in $(\mathrm{km/s})^2$) in the dark matter profile of `node` at the given `radius` (given in units of Mpc) using a numerical calculation.

`double precision radialMomentNumerical(<type((treeNode))> node↔, <double> moment→, <double> radiusMinimum→, <double> radiusMaximum→)` Returns the radial moment of the density in the dark matter profile of `node` between the given `radiusMinimum` and `radiusMaximum` (given in units of Mpc).

`double precision radialVelocityDispersionNumerical(<type((treeNode))> node↔,<double precision> radius`
`argin)` Returns the radial velocity dispersion (in km/s) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

`double precision radiusEnclosingDensityNumerical(<type((treeNode))> node↔,<double precision> density→)` Returns the radius (in Mpc) in the dark matter profile of `node` which encloses the given `density` (given in units of $M_\odot/\mathrm{Mpc}^{-3}$).

`double precision radiusEnclosingMassNumerical(<type((treeNode))> node↔,<double precision> mass→)` Returns the radius (in Mpc) in the dark matter profile of `node` which encloses the given `mass` (given in units of $M_\odot$).

`double precision radiusFromSpecificAngularMomentumNumerical(<type((treeNode))> node↔,<double precision> specificAngularMomentum→)` Returns the radius (in Mpc) in `node` at which a circular orbit has the given `specificAngularMomentum` (given in units of km s^{-1} Mpc).

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`double precision rotationNormalizationNumerical(<type((treeNode))> node↔)` Return the normalization of the rotation velocity vs. specific angular momentum relation.

darkMatterProfileHeatingClass

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((darkMatterProfileHeatingClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`double precision specificEnergy(\textcolor{red}{\textless type((treeNode ))\textgreater} node\arginout,\textcolor{red}{\textless class((darkMatterProfileDMOClass))\textgreater} darkMatterProfileDMO_\arginout,\textcolor{red}{\textless double precision\textgreater} radius\argin)` The specific energy of heating at the given `radius` in the given `node`.

`double precision specificEnergyGradient(\textcolor{red}{\textless type((treeNode ))\textgreater} node\arginout,\textcolor{red}{\textless class((darkMatterProfileDMOClass))\textgreater} darkMatterProfileDMO_\arginout,\textcolor{red}{\textless double precision\textgreater} radius\argin)` The gradient of the specific energy of heating at the given `radius` in the given `node`.

`logical specificEnergyIsEverywhereZero(\textcolor{red}{\textless type((treeNode ))\textgreater} node\arginout,\textcolor{red}{\textless class((darkMatterProfileDMOClass))\textgreater} darkMatterProfileDMO_\arginout)` Returns true if the specific energy is zero everywhere in the given `node`.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

darkMatterProfileHeatingNull

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((darkMatterProfileHeatingClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

double precision specificEnergy(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless class((darkMatterProfileDMOClass))\textgreater} darkMatterProfileDMO_\arginout,\textcolor{red}{\textless double precision\textgreater} radius\argin) The specific energy of heating at the given radius in the given node.

double precision specificEnergyGradient(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless class((darkMatterProfileDMOClass))\textgreater} darkMatterProfileDMO_\arginout,\textcolor{red}{\textless double precision\textgreater} radius\argin) The gradient of the specific energy of heating at the given radius in the given node.

logical specificEnergyIsEverywhereZero(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless class((darkMatterProfileDMOClass))\textgreater} darkMatterProfileDMO_\arginout) Returns true if the specific energy is zero everywhere in the given node.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

darkMatterProfileHeatingSummation

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((darkMatterProfileHeatingClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`double precision specificEnergy(\textcolor{red}{\textless type((treeNode ))\textgreater} node\arginout,\textcolor{red}{\textless class((darkMatterProfileDMOClass))\textgreater} darkMatterProfileDMO_\arginout,\textcolor{red}{\textless double precision\textgreater} radius\argin)` The specific energy of heating at the given `radius` in the given `node`.

`double precision specificEnergyGradient(\textcolor{red}{\textless type((treeNode ))\textgreater} node\arginout,\textcolor{red}{\textless class((darkMatterProfileDMOClass))\textgreater} darkMatterProfileDMO_\arginout,\textcolor{red}{\textless double precision\textgreater} radius\argin)` The gradient of the specific energy of heating at the given `radius` in the given `node`.

`logical specificEnergyIsEverywhereZero(\textcolor{red}{\textless type((treeNode ))\textgreater} node\arginout,\textcolor{red}{\textless class((darkMatterProfileDMOClass))\textgreater} darkMatterProfileDMO_\arginout)` Returns true if the specific energy is zero everywhere in the given `node`.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

darkMatterProfileHeatingTidal

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((darkMatterProfileHeatingClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

double precision specificEnergy(\textcolor{red}{\textless type((treeNode))\textgreater}
node\arginout,\textcolor{red}{\textless class((darkMatterProfileDMOClass))\textgreater}
darkMatterProfileDMO_\arginout,\textcolor{red}{\textless double precision\textgreater}
radius\argin) The specific energy of heating at the given radius in the given node.

double precision specificEnergyGradient(\textcolor{red}{\textless type((treeNode))\textgreater}
node\arginout,\textcolor{red}{\textless class((darkMatterProfileDMOClass))\textgreater}
darkMatterProfileDMO_\arginout,\textcolor{red}{\textless double precision\textgreater}
radius\argin) The gradient of the specific energy of heating at the given radius in the given node.

logical specificEnergyIsEverywhereZero(\textcolor{red}{\textless type((treeNode))\textgreater}
node\arginout,\textcolor{red}{\textless class((darkMatterProfileDMOClass))\textgreater}
darkMatterProfileDMO_\arginout) Returns true if the specific energy is zero everywhere in the given node.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

darkMatterProfileHeatingTwoBodyRelaxation

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((darkMatterProfileHeatingClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`double precision specificEnergy(\textcolor{red}{\textless type((treeNode ))\textgreater} node\arginout,\textcolor{red}{\textless class((darkMatterProfileDMOClass))\textgreater} darkMatterProfileDMO_\arginout,\textcolor{red}{\textless double precision\textgreater} radius\argin)` The specific energy of heating at the given `radius` in the given `node`.

`double precision specificEnergyGradient(\textcolor{red}{\textless type((treeNode ))\textgreater} node\arginout,\textcolor{red}{\textless class((darkMatterProfileDMOClass))\textgreater} darkMatterProfileDMO_\arginout,\textcolor{red}{\textless double precision\textgreater} radius\argin)` The gradient of the specific energy of heating at the given `radius` in the given `node`.

`logical specificEnergyIsEverywhereZero(\textcolor{red}{\textless type((treeNode ))\textgreater} node\arginout,\textcolor{red}{\textless class((darkMatterProfileDMOClass))\textgreater} darkMatterProfileDMO_\arginout)` Returns true if the specific energy is zero everywhere in the given `node`.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

darkMatterProfileScaleRadiusBinary

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((darkMatterProfileScaleRadiusClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision radius(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)
Returns the scale radius for the given node.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

darkMatterProfileScaleRadiusClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((darkMatterProfileScaleRadiusClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision radius(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)
Returns the scale radius for the given node.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

darkMatterProfileScaleRadiusConcentration

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((darkMatterProfileScaleRadiusClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision radius(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)
Returns the scale radius for the given node.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

darkMatterProfileScaleRadiusLudlow2014

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((darkMatterProfileScaleRadiusClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

<double> formationTimeRoot(<double> timeFormation→) Evalute a function which goes to zero at the formation time of the tree.

<void> formationTimeRootFunctionSet(<type(rootFinder)> finder↔) Initialize a root finder object for use in finding the formation time of the tree.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision radius(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the scale radius for the given node.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

darkMatterProfileScaleRadiusLudlow2016

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((darkMatterProfileScaleRadiusClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

<double> formationTimeRoot(<double> timeFormation→) Evalute a function which goes to zero at the formation time of the tree.

<void> formationTimeRootFunctionSet(<type(rootFinder)> finder↔) Initialize a root finder object for use in finding the formation time of the tree.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision radius(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)
Returns the scale radius for the given node.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

darkMatterProfileScaleRadiusZero

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((darkMatterProfileScaleRadiusClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`double precision radius(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Returns the scale radius for the given `node`.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

darkMatterProfileShapeClass

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((darkMatterProfileShapeClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

double precision shape(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the shape parameter for the given node.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

darkMatterProfileShapeGao2008

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((darkMatterProfileShapeClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

double precision shape(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the shape parameter for the given node.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

darkMatterProfileShapeKlypin2015

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((darkMatterProfileShapeClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`double precision shape(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Returns the shape parameter for the given `node`.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

differentiator

`double precision derivative(<double> x→, <double> [h]←, <double> [errorAbsolute]←)` Returns the derivative of the function at argument `x`.

distributionFunction1DBeta

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`double precision cumulative(\textcolor{red}{\textless double precision\textgreater} x\argin)` Return the cumulative probability at `x`.

`void deepCopy(\textcolor{red}{\textless class((distributionFunction1DClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`double precision density(\textcolor{red}{\textless double precision\textgreater} x\argin)` Return the probability density at `x`.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`double precision inverse(\textcolor{red}{\textless double precision\textgreater} p\argin)` Return the value of the independent variable corresponding to cumulative probability `p`.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision maximum()` Returns the maximum possible value in the distribution.

`double precision minimum()` Returns the minimum possible value in the distribution.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`double precision sample(\textcolor{red}{\textless integer\textgreater} incrementSeed\argin,\textcolor{r logical\textgreater} ompThreadOffset\argin,\textcolor{red}{\textless logical\textgreater} mpiRankOffset\argin,\textcolor{red}{\textless type((pseudoRandom ))\textgreater} randomNumberGenera` Return a random deviate from the distribution.

`void samplerReset()` Reset the sampler for the distribution.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

distributionFunction1DCauchy

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`double precision cumulative(\textcolor{red}{\textless double precision\textgreater} x\argin)` Return the cumulative probability at `x`.

void deepCopy(\textcolor{red}{\textless class((distributionFunction1DClass))\textgreater} destination\arginout) Perform a deep copy of the object.

double precision density(\textcolor{red}{\textless double precision\textgreater} x\argin) Return the probability density at x.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

double precision inverse(\textcolor{red}{\textless double precision\textgreater} p\argin) Return the value of the independent variable corresponding to cumulative probability p.

<logical> isDefault() Return true if this is the default object of this class.

double precision maximum() Returns the maximum possible value in the distribution.

double precision minimum() Returns the minimum possible value in the distribution.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

double precision sample(\textcolor{red}{\textless integer\textgreater} incrementSeed\argin,\textcolor{r logical\textgreater} ompThreadOffset\argin,\textcolor{red}{\textless logical\textgreater} mpiRankOffset\argin,\textcolor{red}{\textless type((pseudoRandom))\textgreater} randomNumberGenera Return a random deviate from the distribution.

void samplerReset() Reset the sampler for the distribution.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

distributionFunction1DClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

double precision cumulative(\textcolor{red}{\textless double precision\textgreater} x\argin) Return the cumulative probability at x.

`void deepCopy(\textcolor{red}{\textless class((distributionFunction1DClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`double precision density(\textcolor{red}{\textless double precision\textgreater} x\argin)` Return the probability density at `x`.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`double precision inverse(\textcolor{red}{\textless double precision\textgreater} p\argin)` Return the value of the independent variable corresponding to cumulative probability `p`.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision maximum()` Returns the maximum possible value in the distribution.

`double precision minimum()` Returns the minimum possible value in the distribution.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`double precision sample(\textcolor{red}{\textless integer\textgreater} incrementSeed\argin,\textcolor{r logical\textgreater} ompThreadOffset\argin,\textcolor{red}{\textless logical\textgreater} mpiRankOffset\argin,\textcolor{red}{\textless type((pseudoRandom ))\textgreater} randomNumberGenera` Return a random deviate from the distribution.

`void samplerReset()` Reset the sampler for the distribution.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

distributionFunction1DGamma

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`double precision cumulative(\textcolor{red}{\textless double precision\textgreater} x\argin)` Return the cumulative probability at `x`.

`void deepCopy(\textcolor{red}{\textless class((distributionFunction1DClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`double precision density(\textcolor{red}{\textless double precision\textgreater} x\argin)` Return the probability density at `x`.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`double precision inverse(\textcolor{red}{\textless double precision\textgreater} p\argin)` Return the value of the independent variable corresponding to cumulative probability `p`.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision maximum()` Returns the maximum possible value in the distribution.

`double precision minimum()` Returns the minimum possible value in the distribution.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`double precision sample(\textcolor{red}{\textless integer\textgreater} incrementSeed\argin,\textcolor{r logical\textgreater} ompThreadOffset\argin,\textcolor{red}{\textless logical\textgreater} mpiRankOffset\argin,\textcolor{red}{\textless type((pseudoRandom ))\textgreater} randomNumberGenera` Return a random deviate from the distribution.

`void samplerReset()` Reset the sampler for the distribution.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

distributionFunction1DLogNormal

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`double precision cumulative(\textcolor{red}{\textless double precision\textgreater} x\argin)` Return the cumulative probability at `x`.

`void deepCopy(\textcolor{red}{\textless class((distributionFunction1DClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`double precision density(\textcolor{red}{\textless double precision\textgreater} x\argin)` Return the probability density at `x`.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`double precision inverse(\textcolor{red}{\textless double precision\textgreater} p\argin)` Return the value of the independent variable corresponding to cumulative probability `p`.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision maximum()` Returns the maximum possible value in the distribution.

`double precision minimum()` Returns the minimum possible value in the distribution.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`double precision sample(\textcolor{red}{\textless integer\textgreater} incrementSeed\argin,\textcolor{r logical\textgreater} ompThreadOffset\argin,\textcolor{red}{\textless logical\textgreater} mpiRankOffset\argin,\textcolor{red}{\textless type((pseudoRandom ))\textgreater} randomNumberGenera` Return a random deviate from the distribution.

`void samplerReset()` Reset the sampler for the distribution.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

distributionFunction1DLogUniform

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`double precision cumulative(\textcolor{red}{\textless double precision\textgreater} x\argin)` Return the cumulative probability at `x`.

`void deepCopy(\textcolor{red}{\textless class((distributionFunction1DClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`double precision density(\textcolor{red}{\textless double precision\textgreater} x\argin)` Return the probability density at `x`.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`double precision inverse(\textcolor{red}{\textless double precision\textgreater} p\argin)` Return the value of the independent variable corresponding to cumulative probability `p`.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision maximum()` Returns the maximum possible value in the distribution.

`double precision minimum()` Returns the minimum possible value in the distribution.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`double precision sample(\textcolor{red}{\textless integer\textgreater} incrementSeed\argin,\textcolor{r logical\textgreater} ompThreadOffset\argin,\textcolor{red}{\textless logical\textgreater} mpiRankOffset\argin,\textcolor{red}{\textless type((pseudoRandom ))\textgreater} randomNumberGenera` Return a random deviate from the distribution.

`void samplerReset()` Reset the sampler for the distribution.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

distributionFunction1DNegativeExponential

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`double precision cumulative(\textcolor{red}{\textless double precision\textgreater} x\argin)` Return the cumulative probability at `x`.

`void deepCopy(\textcolor{red}{\textless class((distributionFunction1DClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`double precision density(\textcolor{red}{\textless double precision\textgreater} x\argin)` Return the probability density at `x`.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`double precision inverse(\textcolor{red}{\textless double precision\textgreater} p\argin)` Return the value of the independent variable corresponding to cumulative probability `p`.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision maximum()` Returns the maximum possible value in the distribution.

`double precision minimum()` Returns the minimum possible value in the distribution.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`double precision sample(\textcolor{red}{\textless integer\textgreater} incrementSeed\argin,\textcolor{r logical\textgreater} ompThreadOffset\argin,\textcolor{red}{\textless logical\textgreater} mpiRankOffset\argin,\textcolor{red}{\textless type((pseudoRandom ))\textgreater} randomNumberGenera` Return a random deviate from the distribution.

`void samplerReset()` Reset the sampler for the distribution.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

distributionFunction1DNormal

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`double precision cumulative(\textcolor{red}{\textless double precision\textgreater} x\argin)` Return the cumulative probability at `x`.

`void deepCopy(\textcolor{red}{\textless class((distributionFunction1DClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`double precision density(\textcolor{red}{\textless double precision\textgreater} x\argin)` Return the probability density at `x`.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`double precision inverse(\textcolor{red}{\textless double precision\textgreater} p\argin)` Return the value of the independent variable corresponding to cumulative probability `p`.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision maximum()` Returns the maximum possible value in the distribution.

`double precision minimum()` Returns the minimum possible value in the distribution.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`double precision sample(\textcolor{red}{\textless integer\textgreater} incrementSeed\argin,\textcolor{r logical\textgreater} ompThreadOffset\argin,\textcolor{red}{\textless logical\textgreater} mpiRankOffset\argin,\textcolor{red}{\textless type((pseudoRandom ))\textgreater} randomNumberGenera` Return a random deviate from the distribution.

`void samplerReset()` Reset the sampler for the distribution.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

distributionFunction1DPeakBackground

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`double precision cumulative(\textcolor{red}{\textless double precision\textgreater} x\argin)` Return the cumulative probability at `x`.

`void deepCopy(\textcolor{red}{\textless class((distributionFunction1DClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`double precision density(\textcolor{red}{\textless double precision\textgreater} x\argin)` Return the probability density at `x`.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`double precision inverse(\textcolor{red}{\textless double precision\textgreater} p\argin)` Return the value of the independent variable corresponding to cumulative probability `p`.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision maximum()` Returns the maximum possible value in the distribution.

`double precision minimum()` Returns the minimum possible value in the distribution.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`double precision sample(\textcolor{red}{\textless integer\textgreater} incrementSeed\argin,\textcolor{r logical\textgreater} ompThreadOffset\argin,\textcolor{red}{\textless logical\textgreater} mpiRankOffset\argin,\textcolor{red}{\textless type((pseudoRandom ))\textgreater} randomNumberGenera` Return a random deviate from the distribution.

`void samplerReset()` Reset the sampler for the distribution.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

distributionFunction1DStudentT

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`double precision cumulative(\textcolor{red}{\textless double precision\textgreater} x\argin)` Return the cumulative probability at `x`.

void deepCopy(\textcolor{red}{\textless class((distributionFunction1DClass))\textgreater} destination\arginout) Perform a deep copy of the object.

double precision density(\textcolor{red}{\textless double precision\textgreater} x\argin) Return the probability density at x.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

double precision inverse(\textcolor{red}{\textless double precision\textgreater} p\argin) Return the value of the independent variable corresponding to cumulative probability p.

<logical> isDefault() Return true if this is the default object of this class.

double precision maximum() Returns the maximum possible value in the distribution.

double precision minimum() Returns the minimum possible value in the distribution.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

double precision sample(\textcolor{red}{\textless integer\textgreater} incrementSeed\argin,\textcolor{r logical\textgreater} ompThreadOffset\argin,\textcolor{red}{\textless logical\textgreater} mpiRankOffset\argin,\textcolor{red}{\textless type((pseudoRandom))\textgreater} randomNumberGenera Return a random deviate from the distribution.

void samplerReset() Reset the sampler for the distribution.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

distributionFunction1DUniform

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

double precision cumulative(\textcolor{red}{\textless double precision\textgreater} x\argin) Return the cumulative probability at x.

`void deepCopy(\textcolor{red}{\textless class((distributionFunction1DClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`double precision density(\textcolor{red}{\textless double precision\textgreater} x\argin)` Return the probability density at `x`.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`double precision inverse(\textcolor{red}{\textless double precision\textgreater} p\argin)` Return the value of the independent variable corresponding to cumulative probability `p`.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision maximum()` Returns the maximum possible value in the distribution.

`double precision minimum()` Returns the minimum possible value in the distribution.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`double precision sample(\textcolor{red}{\textless integer\textgreater} incrementSeed\argin,\textcolor{r logical\textgreater} ompThreadOffset\argin,\textcolor{red}{\textless logical\textgreater} mpiRankOffset\argin,\textcolor{red}{\textless type((pseudoRandom ))\textgreater} randomNumberGenera` Return a random deviate from the distribution.

`void samplerReset()` Reset the sampler for the distribution.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

distributionFunction1DVoight

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`double precision cumulative(\textcolor{red}{\textless double precision\textgreater} x\argin)` Return the cumulative probability at `x`.

`void deepCopy(\textcolor{red}{\textless class((distributionFunction1DClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`double precision density(\textcolor{red}{\textless double precision\textgreater} x\argin)` Return the probability density at `x`.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`double precision inverse(\textcolor{red}{\textless double precision\textgreater} p\argin)` Return the value of the independent variable corresponding to cumulative probability `p`.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision maximum()` Returns the maximum possible value in the distribution.

`double precision minimum()` Returns the minimum possible value in the distribution.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`double precision sample(\textcolor{red}{\textless integer\textgreater} incrementSeed\argin,\textcolor{r logical\textgreater} ompThreadOffset\argin,\textcolor{red}{\textless logical\textgreater} mpiRankOffset\argin,\textcolor{red}{\textless type((pseudoRandom ))\textgreater} randomNumberGenera` Return a random deviate from the distribution.

`void samplerReset()` Reset the sampler for the distribution.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

distributionFunctionDiscrete1DBinomial

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`double precision cumulative(\textcolor{red}{\textless integer\textgreater} x\argin)` Return the cumulative probability at `x`.

`void deepCopy(\textcolor{red}{\textless class((distributionFunctionDiscrete1DClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`integer inverse(\textcolor{red}{\textless double precision\textgreater} p\argin)` Return the value of the independent variable corresponding to cumulative probability `p`.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision mass(\textcolor{red}{\textless integer\textgreater} x\argin)` Return the probability mass at `x`.

`double precision massLogarithmic(\textcolor{red}{\textless integer\textgreater} x\argin)` Return the logarithm of the probability mass at `x`.

`integer maximum()` Returns the maximum possible value in the distribution.

`integer minimum()` Returns the minimum possible value in the distribution.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`integer sample(\textcolor{red}{\textless integer\textgreater} incrementSeed\argin,\textcolor{red}{\text logical\textgreater} ompThreadOffset\argin,\textcolor{red}{\textless logical\textgreater} mpiRankOffset\argin,\textcolor{red}{\textless type((pseudoRandom ))\textgreater} randomNumberGenera` Return a random deviate from the distribution.

`void samplerReset()` Reset the sampler for the distribution.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

distributionFunctionDiscrete1DClass

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`double precision cumulative(\textcolor{red}{\textless integer\textgreater} x\argin)` Return the cumulative probability at `x`.

`void deepCopy(\textcolor{red}{\textless class((distributionFunctionDiscrete1DClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`integer inverse(\textcolor{red}{\textless double precision\textgreater} p\argin)` Return the value of the independent variable corresponding to cumulative probability `p`.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision mass(\textcolor{red}{\textless integer\textgreater} x\argin)` Return the probability mass at `x`.

`double precision massLogarithmic(\textcolor{red}{\textless integer\textgreater} x\argin)` Return the logarithm of the probability mass at `x`.

`integer maximum()` Returns the maximum possible value in the distribution.

`integer minimum()` Returns the minimum possible value in the distribution.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`integer sample(\textcolor{red}{\textless integer\textgreater} incrementSeed\argin,\textcolor{red}{\text logical\textgreater} ompThreadOffset\argin,\textcolor{red}{\textless logical\textgreater} mpiRankOffset\argin,\textcolor{red}{\textless type((pseudoRandom ))\textgreater} randomNumberGenera` Return a random deviate from the distribution.

`void samplerReset()` Reset the sampler for the distribution.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

distributionFunctionDiscrete1DNegativeBinomial

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`double precision cumulative(\textcolor{red}{\textless integer\textgreater} x\argin)` Return the cumulative probability at `x`.

`void deepCopy(\textcolor{red}{\textless class((distributionFunctionDiscrete1DClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`integer inverse(\textcolor{red}{\textless double precision\textgreater} p\argin)` Return the value of the independent variable corresponding to cumulative probability `p`.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision mass(\textcolor{red}{\textless integer\textgreater} x\argin)` Return the probability mass at `x`.

`double precision massLogarithmic(\textcolor{red}{\textless integer\textgreater} x\argin)` Return the logarithm of the probability mass at `x`.

`integer maximum()` Returns the maximum possible value in the distribution.

`integer minimum()` Returns the minimum possible value in the distribution.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`integer sample(\textcolor{red}{\textless integer\textgreater} incrementSeed\argin,\textcolor{red}{\text logical\textgreater} ompThreadOffset\argin,\textcolor{red}{\textless logical\textgreater} mpiRankOffset\argin,\textcolor{red}{\textless type((pseudoRandom ))\textgreater} randomNumberGenera` Return a random deviate from the distribution.

`void samplerReset()` Reset the sampler for the distribution.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

doubleScalarHash

<void> delete(<(character(len=*)|varying_string)> key→, <integer> value→) Delete a key from the hash.

void destroy() Destroy the hash.

<logical> exists(<(character(len=*)|varying_string|<integer>)> key→) Return true if the specified key exists in the hash.

<void> initialize() Initialize the hash.

<type(varying_string)> key(<integer> indexValue→) Return the key of the indexValue$^{\text{th}}$ entry in the hash.

<void> keys(<type(varying_string)[:]> keys↔) Return an array of all keys in the hash.

<void> set(<(character(len=*)|varying_string)> key→, <integer> value→) Set the value of a key in the hash.

<integer> size() Return the number of keys in the hash.

<integer> value(<(character(len=*)|varying_string|<integer>)> key→, <integer> value→) Return the value for the given key.

<void> values(<double precision[:]> values↔) Return an array of all values in the hash.

eventHook

<integer> count() Return a count of the number of hooks into this event.

HASH(0x1c7ece8) first(<*type(hook)>) Return a pointer to the first hook into this event.

HASH(0x1c83d80) initialize() Initialize the event.

void lock() Lock the event.

void unlock() Unlock the event.

eventHook88992e2a75a1ab33893e7a1e2bf92b3d

<void> attach(<class(*)> *object_→, <procedure()> *function_→) Attach a hook to the event.

<integer> count() Return a count of the number of hooks into this event.

<void> detach(<class(*)> *object_→, <procedure()> *function_→) Detach a hook from the event.

HASH(0x1c7ece8) first(<*type(hook)>) Return a pointer to the first hook into this event.

HASH(0x1c83d80) initialize() Initialize the event.

void lock() Lock the event.

void unlock() Unlock the event.

eventHookUnspecified

<void> attach(<class(*)> *object_→, <procedure()> *function_→) Attach a hook to the event.

<integer> count() Return a count of the number of hooks into this event.

<void> detach(<class(*)> *object_→, <procedure()> *function_→) Detach a hook from the event.

HASH(0x1c7ece8) first(<*type(hook)>) Return a pointer to the first hook into this event.

HASH(0x1c83d80) initialize() Initialize the event.

void lock() Lock the event.

void unlock() Unlock the event.

evolveForestsWorkShareClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((evolveForestsWorkShareClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

integer(c_size_t) forestNumber(\textcolor{red}{\textless logical\textgreater} utilizeOpenMPThreads\argi
Return the number of the forest to process.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

integer workerCount(\textcolor{red}{\textless logical\textgreater} utilizeOpenMPThreads\argin) Return the count of workers.

integer workerID(\textcolor{red}{\textless logical\textgreater} utilizeOpenMPThreads\argin) Return a unique worker ID.

evolveForestsWorkShareCyclic

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((evolveForestsWorkShareClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

integer(c_size_t) forestNumber(\textcolor{red}{\textless logical\textgreater} utilizeOpenMPThreads\argi Return the number of the forest to process.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

integer workerCount(\textcolor{red}{\textless logical\textgreater} utilizeOpenMPThreads\argin) Return the count of workers.

integer workerID(\textcolor{red}{\textless logical\textgreater} utilizeOpenMPThreads\argin) Return a unique worker ID.

evolveForestsWorkShareFCFS

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((evolveForestsWorkShareClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

integer(c_size_t) forestNumber(\textcolor{red}{\textless logical\textgreater} utilizeOpenMPThreads\argi
Return the number of the forest to process.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

integer workerCount(\textcolor{red}{\textless logical\textgreater} utilizeOpenMPThreads\argin)
Return the count of workers.

integer workerID(\textcolor{red}{\textless logical\textgreater} utilizeOpenMPThreads\argin)
Return a unique worker ID.

evolveForestsWorkShareStride

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((evolveForestsWorkShareClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`integer(c_size_t) forestNumber(\textcolor{red}{\textless logical\textgreater} utilizeOpenMPThreads\argi` Return the number of the forest to process.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`integer workerCount(\textcolor{red}{\textless logical\textgreater} utilizeOpenMPThreads\argin)` Return the count of workers.

`integer workerID(\textcolor{red}{\textless logical\textgreater} utilizeOpenMPThreads\argin)` Return a unique worker ID.

excursionSetBarrierClass

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`double precision barrier(\textcolor{red}{\textless double precision\textgreater} variance\argin,\textco double precision\textgreater} time\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless logical\textgreater} rateCompute\argin)` Return the barrier height at the given variance and time. The `rateCompute` should be set to `true` if the barrier is being used in a calculation of barrier crossing rates, and to `false` otherwise.

`double precision barrierGradient(\textcolor{red}{\textless double precision\textgreater} variance\argin,\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\ type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless logical\textgreater} rateCompute\argin)` Return the gradient of the barrier with respect to variance at the given variance and time. The `rateCompute` should be set to `true` if the barrier is being used in a calculation of barrier crossing rates, and to `false` otherwise.

`void deepCopy(\textcolor{red}{\textless class((excursionSetBarrierClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

excursionSetBarrierCriticalOverdensity

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`double precision barrier(\textcolor{red}{\textless double precision\textgreater} variance\argin,\textco double precision\textgreater} time\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless logical\textgreater} rateCompute\argin)` Return the barrier height at the given variance and time. The `rateCompute` should be set to `true` if the barrier is being used in a calculation of barrier crossing rates, and to `false` otherwise.

`double precision barrierGradient(\textcolor{red}{\textless double precision\textgreater} variance\argin,\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\ type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless logical\textgreater} rateCompute\argin)` Return the gradient of the barrier with respect to variance at the given variance and time. The `rateCompute` should be set to `true` if the barrier is being used in a calculation of barrier crossing rates, and to `false` otherwise.

`void deepCopy(\textcolor{red}{\textless class((excursionSetBarrierClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

excursionSetBarrierLinear

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

double precision barrier(\textcolor{red}{\textless double precision\textgreater} variance\argin,\textco
double precision\textgreater} time\argin,\textcolor{red}{\textless type((treeNode))\textgreater}
node\arginout,\textcolor{red}{\textless logical\textgreater} rateCompute\argin) Return the barrier height at the given variance and time. The rateCompute should be set to true if the barrier is being used in a calculation of barrier crossing rates, and to false otherwise.

double precision barrierGradient(\textcolor{red}{\textless double precision\textgreater}
variance\argin,\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\
type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless logical\textgreater}
rateCompute\argin) Return the gradient of the barrier with respect to variance at the given variance and time. The rateCompute should be set to true if the barrier is being used in a calculation of barrier crossing rates, and to false otherwise.

void deepCopy(\textcolor{red}{\textless class((excursionSetBarrierClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

excursionSetBarrierQuadratic

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

double precision barrier(\textcolor{red}{\textless double precision\textgreater} variance\argin,\textco double precision\textgreater} time\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless logical\textgreater} rateCompute\argin) Return the barrier height at the given variance and time. The rateCompute should be set to true if the barrier is being used in a calculation of barrier crossing rates, and to false otherwise.

double precision barrierGradient(\textcolor{red}{\textless double precision\textgreater} variance\argin,\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\ type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless logical\textgreater} rateCompute\argin) Return the gradient of the barrier with respect to variance at the given variance and time. The rateCompute should be set to true if the barrier is being used in a calculation of barrier crossing rates, and to false otherwise.

void deepCopy(\textcolor{red}{\textless class((excursionSetBarrierClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

excursionSetBarrierRemapScale

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

double precision barrier(\textcolor{red}{\textless double precision\textgreater} variance\argin,\textco double precision\textgreater} time\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless logical\textgreater} rateCompute\argin) Return the barrier height at the given variance and time. The rateCompute should be set to true if the barrier is being used in a calculation of barrier crossing rates, and to false otherwise.

double precision barrierGradient(\textcolor{red}{\textless double precision\textgreater} variance\argin,\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\ type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless logical\textgreater} rateCompute\argin) Return the gradient of the barrier with respect to variance at the given variance and time. The rateCompute should be set to true if the barrier is being used in a calculation of barrier crossing rates, and to false otherwise.

void deepCopy(\textcolor{red}{\textless class((excursionSetBarrierClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

excursionSetBarrierRemapShethMoTormen

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`double precision barrier(\textcolor{red}{\textless double precision\textgreater} variance\argin,\textcc double precision\textgreater} time\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless logical\textgreater} rateCompute\argin)` Return the barrier height at the given variance and time. The `rateCompute` should be set to `true` if the barrier is being used in a calculation of barrier crossing rates, and to `false` otherwise.

`double precision barrierGradient(\textcolor{red}{\textless double precision\textgreater} variance\argin,\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\ type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless logical\textgreater} rateCompute\argin)` Return the gradient of the barrier with respect to variance at the given variance and time. The `rateCompute` should be set to `true` if the barrier is being used in a calculation of barrier crossing rates, and to `false` otherwise.

`void deepCopy(\textcolor{red}{\textless class((excursionSetBarrierClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

excursionSetFirstCrossingClass

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void coordinatedMPI(\textcolor{red}{\textless logical\textgreater} state\argin)` Sets the state of coordination under MPI. If set to true then the object can assume that any calculations it performs are being performed identically by all other MPI processes. This permits, for example, coordinated tabulation of results across MPI processes.

`void deepCopy(\textcolor{red}{\textless class((excursionSetFirstCrossingClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`double precision probability(\textcolor{red}{\textless double precision\textgreater} variance\argin,\te double precision\textgreater} time\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Return the probability for a trajectory to make its first crossing of the barrier at the given `variance` and `time`.

`double precision rate(\textcolor{red}{\textless double precision\textgreater} variance\argin,\textcolor double precision\textgreater} varianceProgenitor\argin,\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Return the rate of first crossing for excursion sets beginning at the given `variance` and `time` to transition to a first crossing at the given `varianceProgenitor`.

`double precision rateNonCrossing(\textcolor{red}{\textless double precision\textgreater} variance\argin,\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\ type((treeNode))\textgreater} node\arginout)` Return the rate of non-crossing for excursion sets beginning at the given `variance` and `time`.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

excursionSetFirstCrossingFarahi

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void coordinatedMPI(\textcolor{red}{\textless logical\textgreater} state\argin)` Sets the state of coordination under MPI. If set to true then the object can assume that any calculations it performs are being performed identically by all other MPI processes. This permits, for example, coordinated tabulation of results across MPI processes.

`void deepCopy(\textcolor{red}{\textless class((excursionSetFirstCrossingClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`<void> fileRead()` Read excursion set solutions from file.

`<void> fileWrite()` Write excursion set solutions to file.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`double precision probability(\textcolor{red}{\textless double precision\textgreater} variance\argin,\te double precision\textgreater} time\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Return the probability for a trajectory to make its first crossing of the barrier at the given `variance` and `time`.

`double precision rate(\textcolor{red}{\textless double precision\textgreater} variance\argin,\textcolor double precision\textgreater} varianceProgenitor\argin,\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Return the rate of first crossing for excursion sets beginning at the given `variance` and `time` to transition to a first crossing at the given `varianceProgenitor`.

`double precision rateNonCrossing(\textcolor{red}{\textless double precision\textgreater} variance\argin,\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\ type((treeNode))\textgreater} node\arginout)` Return the rate of non-crossing for excursion sets beginning at the given `variance` and `time`.

`<void> rateTabulate(<double> varianceProgenitor→, <double> time→)` Tabulate excursion set barrier crossing rates ensuring that they span the given progenitor variance and time.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

<double(:)> varianceRange(<double> rangeMinimum→, <double> rangeMaximum→, <integer> rangeNumber, <double> exponent) Build a range of variances at which to tabulate the excursion set solutions.

excursionSetFirstCrossingFarahiMidpoint

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void coordinatedMPI(\textcolor{red}{\textless logical\textgreater} state\argin) Sets the state of coordination under MPI. If set to true then the object can assume that any calculations it performs are being performed identically by all other MPI processes. This permits, for example, coordinated tabulation of results across MPI processes.

void deepCopy(\textcolor{red}{\textless class((excursionSetFirstCrossingClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

<void> fileRead() Read excursion set solutions from file.

<void> fileWrite() Write excursion set solutions to file.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision probability(\textcolor{red}{\textless double precision\textgreater} variance\argin,\te double precision\textgreater} time\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Return the probability for a trajectory to make its first crossing of the barrier at the given variance and time.

double precision rate(\textcolor{red}{\textless double precision\textgreater} variance\argin,\textcolor double precision\textgreater} varianceProgenitor\argin,\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Return the rate of first crossing for excursion sets beginning at the given variance and time to transition to a first crossing at the given varianceProgenitor.

double precision rateNonCrossing(\textcolor{red}{\textless double precision\textgreater} variance\argin,\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\type((treeNode))\textgreater} node\arginout) Return the rate of non-crossing for excursion sets beginning at the given variance and time.

<void> rateTabulate(<double> varianceProgenitor→, <double> time→) Tabulate excursion set barrier crossing rates ensuring that they span the given progenitor variance and time.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

<double(:)> varianceRange(<double> rangeMinimum→, <double> rangeMaximum→, <integer> rangeNumber, <double> exponent) Build a range of variances at which to tabulate the excursion set solutions.

excursionSetFirstCrossingLinearBarrier

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void coordinatedMPI(\textcolor{red}{\textless logical\textgreater} state\argin) Sets the state of coordination under MPI. If set to true then the object can assume that any calculations it performs are being performed identically by all other MPI processes. This permits, for example, coordinated tabulation of results across MPI processes.

void deepCopy(\textcolor{red}{\textless class((excursionSetFirstCrossingClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision probability(\textcolor{red}{\textless double precision\textgreater} variance\argin,\te double precision\textgreater} time\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Return the probability for a trajectory to make its first crossing of the barrier at the given variance and time.

double precision rate(\textcolor{red}{\textless double precision\textgreater} variance\argin,\textcolor double precision\textgreater} varianceProgenitor\argin,\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Return the rate of first crossing for excursion sets beginning at the given variance and time to transition to a first crossing at the given varianceProgenitor.

double precision rateNonCrossing(\textcolor{red}{\textless double precision\textgreater} variance\argin,\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\ type((treeNode))\textgreater} node\arginout) Return the rate of non-crossing for excursion sets beginning at the given variance and time.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

excursionSetFirstCrossingZhangHui

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void coordinatedMPI(\textcolor{red}{\textless logical\textgreater} state\argin) Sets the state of coordination under MPI. If set to true then the object can assume that any calculations it performs are being performed identically by all other MPI processes. This permits, for example, coordinated tabulation of results across MPI processes.

void deepCopy(\textcolor{red}{\textless class((excursionSetFirstCrossingClass))\textgreater} destination\arginout) Perform a deep copy of the object.

<double> delta(<integer> i→, <integer> j→,<double> iVariance→, <double> jVariance→, <double> deltaVariance→, <double> time→) Returns the function $g_2(S, S')$ integrated over a range ΔS [Zhang and Hui, 2006].

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

<double> g1(<double> variance→, <double> time→) Returns the function $g_1(S)$ [Zhang and Hui, 2006].

<double> g2(<double> variance→, <double> variancePrimed→, <double> time→) Returns the function $g_2(S, S')$ [Zhang and Hui, 2006].

<double> g2Integrated(<double> variance→, <double> deltaVariance→, <double> time→) Returns the function $g_2(S, S')$ integrated over a range ΔS [Zhang and Hui, 2006].

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision probability(\textcolor{red}{\textless double precision\textgreater} variance\argin,\te double precision\textgreater} time\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Return the probability for a trajectory to make its first crossing of the barrier at the given variance and time.

double precision rate(\textcolor{red}{\textless double precision\textgreater} variance\argin,\textcolor double precision\textgreater} varianceProgenitor\argin,\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Return the rate of first crossing for excursion sets beginning at the given variance and time to transition to a first crossing at the given varianceProgenitor.

double precision rateNonCrossing(\textcolor{red}{\textless double precision\textgreater} variance\argin,\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\ type((treeNode))\textgreater} node\arginout) Return the rate of non-crossing for excursion sets beginning at the given variance and time.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

excursionSetFirstCrossingZhangHuiHighOrder

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

`void coordinatedMPI(\textcolor{red}{\textless logical\textgreater} state\argin)` Sets the state of coordination under MPI. If set to true then the object can assume that any calculations it performs are being performed identically by all other MPI processes. This permits, for example, coordinated tabulation of results across MPI processes.

`void deepCopy(\textcolor{red}{\textless class((excursionSetFirstCrossingClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`<double> delta(<integer> i→, <integer> j→,<double> iVariance→, <double> jVariance→, <double> deltaVariance→, <double> time→)` Returns the function $g_2(S, S')$ integrated over a range ΔS [Zhang and Hui, 2006].

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`<double> g1(<double> variance→, <double> time→)` Returns the function $g_1(S)$ [Zhang and Hui, 2006].

`<double> g2(<double> variance→, <double> variancePrimed→, <double> time→)` Returns the function $g_2(S, S')$ [Zhang and Hui, 2006].

`<double> g2Integrated(<double> variance→, <double> deltaVariance→, <double> time→)` Returns the function $g_2(S, S')$ integrated over a range ΔS [Zhang and Hui, 2006].

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<void> initialize()` Initialize the high order Zhang and Hui [2006] class.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`double precision probability(\textcolor{red}{\textless double precision\textgreater} variance\argin,\te double precision\textgreater} time\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Return the probability for a trajectory to make its first crossing of the barrier at the given `variance` and `time`.

`double precision rate(\textcolor{red}{\textless double precision\textgreater} variance\argin,\textcolor double precision\textgreater} varianceProgenitor\argin,\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Return the rate of first crossing for excursion sets beginning at the given `variance` and `time` to transition to a first crossing at the given `varianceProgenitor`.

`double precision rateNonCrossing(\textcolor{red}{\textless double precision\textgreater} variance\argin,\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\ type((treeNode))\textgreater} node\arginout)` Return the rate of non-crossing for excursion sets beginning at the given `variance` and `time`.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

fastExponentiator

<double> exponentiate(<double> x→) Evaluate x^y using table look-up.

freefallRadiusClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((freefallRadiusClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision radius(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the freefall radius for gas in the hot atmosphere surrounding the galaxy in node in units of Mpc.

double precision radiusGrowthRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the rate of increase of the freefall radius for gas in the hot atmosphere surrounding the galaxy in node in units of Mpc/Gyr.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

freefallRadiusDarkMatterHalo

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((freefallRadiusClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision radius(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the freefall radius for gas in the hot atmosphere surrounding the galaxy in node in units of Mpc.

double precision radiusGrowthRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the rate of increase of the freefall radius for gas in the hot atmosphere surrounding the galaxy in node in units of Mpc/Gyr.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

freefallTimeAvailableClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((freefallTimeAvailableClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision timeAvailable(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the time available for freefall in cooling calculations in node.

double precision timeAvailableIncreaseRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the rate at which the time available for freefall in cooling calculations increases in node.

freefallTimeAvailableHaloFormation

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((freefallTimeAvailableClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision timeAvailable(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the time available for freefall in cooling calculations in node.

double precision timeAvailableIncreaseRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the rate at which the time available for freefall in cooling calculations increases in node.

functionClass

<logical> isDefault() Return true if this is the default object of this class.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

galacticDynamicsBarInstabilityClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((galacticDynamicsBarInstabilityClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

void timescale(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\t double precision\textgreater} timescale\argout,\textcolor{red}{\textless double precision\textgreat externalDrivingSpecificTorque\argout) Returns a timescale on which the bar instability depletes material from a disk into a pseudo-bulge. A negative value indicates no instability. Also returns the net torque due to any external force causing this instability.

galacticDynamicsBarInstabilityEfstathiou1982

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((galacticDynamicsBarInstabilityClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

<double> estimator() Compute the stability estimator for the Efstathiou et al. [1982] model for galactic disk bar instability.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`void timescale(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\t double precision\textgreater} timescale\argout,\textcolor{red}{\textless double precision\textgreat externalDrivingSpecificTorque\argout)` Returns a timescale on which the bar instability depletes material from a disk into a pseudo-bulge. A negative value indicates no instability. Also returns the net torque due to any external force causing this instability.

galacticDynamicsBarInstabilityEfstathiou1982Tidal

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((galacticDynamicsBarInstabilityClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`<double> estimator()` Compute the stability estimator for the Efstathiou et al. [1982] model for galactic disk bar instability.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`<double> tidalTensorRadial()` Compute the radial term of the tidal tensor.

`void timescale(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\t double precision\textgreater} timescale\argout,\textcolor{red}{\textless double precision\textgreat externalDrivingSpecificTorque\argout)` Returns a timescale on which the bar instability depletes material from a disk into a pseudo-bulge. A negative value indicates no instability. Also returns the net torque due to any external force causing this instability.

galacticDynamicsBarInstabilityFixedTimescale

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((galacticDynamicsBarInstabilityClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

void timescale(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\t
double precision\textgreater} timescale\argout,\textcolor{red}{\textless double precision\textgreat
externalDrivingSpecificTorque\argout) Returns a timescale on which the bar instability depletes material from a disk into a pseudo-bulge. A negative value indicates no instability. Also returns the net torque due to any external force causing this instability.

galacticDynamicsBarInstabilityStable

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((galacticDynamicsBarInstabilityClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`void timescale(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\t double precision\textgreater} timescale\argout,\textcolor{red}{\textless double precision\textgreat externalDrivingSpecificTorque\argout)` Returns a timescale on which the bar instability depletes material from a disk into a pseudo-bulge. A negative value indicates no instability. Also returns the net torque due to any external force causing this instability.

galacticFilterAll

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((galacticFilterClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`logical passes(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Return true if the given `node` passes the filter.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

galacticFilterAlways

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((galacticFilterClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

logical passes(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Return true if the given node passes the filter.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

galacticFilterAny

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((galacticFilterClass))\textgreater} destination\arginout)
Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

logical passes(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Return true if the given node passes the filter.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

galacticFilterBasicMass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((galacticFilterClass))\textgreater} destination\arginout)
Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

logical passes(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Return true if the given node passes the filter.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

galacticFilterClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((galacticFilterClass))\textgreater} destination\arginout)
Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

logical passes(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Return true if the given node passes the filter.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

galacticFilterFormationTime

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((galacticFilterClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

logical passes(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Return true if the given node passes the filter.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

galacticFilterHaloIsolated

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((galacticFilterClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`logical passes(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Return true if the given `node` passes the filter.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

galacticFilterHaloMass

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((galacticFilterClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`logical passes(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Return true if the given `node` passes the filter.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

galacticFilterHaloNotIsolated

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((galacticFilterClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

logical passes(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Return true if the given node passes the filter.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

galacticFilterHostMassRange

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((galacticFilterClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`logical passes(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Return true if the given `node` passes the filter.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

galacticFilterISMMass

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((galacticFilterClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

logical passes(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Return true if the given node passes the filter.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

galacticFilterLightcone

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((galacticFilterClass))\textgreater} destination\arginout)
Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

logical passes(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Return true if the given node passes the filter.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

galacticFilterMainBranch

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((galacticFilterClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`logical passes(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Return true if the given `node` passes the filter.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

galacticFilterNodeMajorMergerRecent

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((galacticFilterClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`logical passes(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Return true if the given `node` passes the filter.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

galacticFilterNot

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((galacticFilterClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`logical passes(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Return true if the given `node` passes the filter.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

galacticFilterRootNode

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((galacticFilterClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

logical passes(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Return true if the given node passes the filter.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

galacticFilterSpheroidStellarMass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((galacticFilterClass))\textgreater} destination\arginout)
Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

logical passes(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Return true if the given node passes the filter.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

galacticFilterStarFormationRate

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((galacticFilterClass))\textgreater} destination\arginout)
Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

logical passes(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Return true if the given node passes the filter.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

galacticFilterStellarAbsoluteMagnitudes

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((galacticFilterClass))\textgreater} destination\arginout)
Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

logical passes(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Return true if the given node passes the filter.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

galacticFilterStellarApparentMagnitudes

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((galacticFilterClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`logical passes(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Return true if the given `node` passes the filter.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

galacticFilterStellarMass

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((galacticFilterClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`logical passes(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Return true if the given `node` passes the filter.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

galacticFilterStellarMassMorphology

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((galacticFilterClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`logical passes(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Return true if the given `node` passes the filter.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

galacticFilterSurveyGeometry

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((galacticFilterClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

logical passes(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Return true if the given node passes the filter.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

galacticFilterTreeHosted

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((galacticFilterClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`logical passes(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Return true if the given `node` passes the filter.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

galacticStructureSolverClass

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((galacticStructureSolverClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void revert(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Revert the structure of components in the given node (if necessary to ensure that the structure solver will give the same result when called consecutively).

void solve(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Solves for the structure of components in the given node.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

galacticStructureSolverEquilibrium

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((galacticStructureSolverClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void revert(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Revert the structure of components in the given node (if necessary to ensure that the structure solver will give the same result when called consecutively).

void solve(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Solves for the structure of components in the given node.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

galacticStructureSolverFixed

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((galacticStructureSolverClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void revert(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Revert the structure of components in the given node (if necessary to ensure that the structure solver will give the same result when called consecutively).

void solve(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Solves for the structure of components in the given node.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

galacticStructureSolverLinear

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((galacticStructureSolverClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void revert(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Revert the structure of components in the given node (if necessary to ensure that the structure solver will give the same result when called consecutively).

void solve(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Solves for the structure of components in the given node.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

galacticStructureSolverSimple

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((galacticStructureSolverClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void revert(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Revert the structure of components in the given `node` (if necessary to ensure that the structure solver will give the same result when called consecutively).

`void solve(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Solves for the structure of components in the given `node`.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

gauntFactorClass

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((gauntFactorClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision total(\textcolor{red}{\textless integer\textgreater} atomicNumber\argin,\textcolor{red integer\textgreater} electronNumber\argin,\textcolor{red}{\textless double precision\textgreater} temperature\argin) Returns the thermally averaged, total Gaunt factor.

gauntFactorSutherland1998

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((gauntFactorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision total(\textcolor{red}{\textless integer\textgreater} atomicNumber\argin,\textcolor{red integer\textgreater} electronNumber\argin,\textcolor{red}{\textless double precision\textgreater} temperature\argin) Returns the thermally averaged, total Gaunt factor.

gauntFactorVanHoof2014

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((gauntFactorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision total(\textcolor{red}{\textless integer\textgreater} atomicNumber\argin,\textcolor{red integer\textgreater} electronNumber\argin,\textcolor{red}{\textless double precision\textgreater} temperature\argin) Returns the thermally averaged, total Gaunt factor.

genericScalarHash

<void> delete(<(character(len=*)|varying_string)> key→, <integer> value→) Delete a key from the hash.

void destroy() Destroy the hash.

<logical> exists(<(character(len=*)|varying_string|<integer>)> key→) Return true if the specified key exists in the hash.

<void> initialize() Initialize the hash.

<type(varying_string)> key(<integer> indexValue→) Return the key of the indexValue$^{\text{th}}$ entry in the hash.

<void> keys(<type(varying_string)[:]> keys↔) Return an array of all keys in the hash.

<void> set(<(character(len=*)|varying_string)> key→, <integer> value→) Set the value of a key in the hash.

<integer> size() Return the number of keys in the hash.

<integer> value(<(character(len=*)|varying_string|<integer>)> key→, <integer> value→) Return the value for the given key.

<void> values(<class(*), pointer[:]> values↔) Return an array of all values in the hash.

geometryLightconeClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((geometryLightconeClass))\textgreater} destination\argino Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

logical isInLightcone(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{ logical\textgreater} atPresentEpoch\argin,\textcolor{red}{\textless double precision\textgreater} radiusBuffer\argin) Returns true if the provided node lies within the lightcone.

type(varying_string) objectType() Return the type of the object.

double precision, dimension(3) position(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer((c_size_t))\textgreater} instance\argin) Returns the position vector of a node (in units of Mpc) in the lightcone coordinate system.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

integer(c_size_t) replicationCount(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the number of times the given nodes appears in the lightone .

double precision solidAngle() Returns the solid angle subtended by the lightcone (in units of steradians).

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision, dimension(3) velocity(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer((c_size_t))\textgreater} instance\argin)` Returns the velocity vector of a `node` (in units of km/s) in the lightcone coordinate system.

geometryLightconeSquare

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((geometryLightconeClass))\textgreater} destination\argino` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`logical isInLightcone(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{ logical\textgreater} atPresentEpoch\argin,\textcolor{red}{\textless double precision\textgreater} radiusBuffer\argin)` Returns true if the provided node lies within the lightcone.

`type(varying_string) objectType()` Return the type of the object.

`double precision, dimension(3) position(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer((c_size_t))\textgreater} instance\argin)` Returns the position vector of a `node` (in units of Mpc) in the lightcone coordinate system.

`<double(3)> positionAtOutput(<integer> output→, <double(3)> nodePosition→, <logical> [positionFound]←)` Returns the position of a point, `nodePosition` (given in physical coordinates within the primary replicant volume), in comoving coordinates in the replicant volume in which it appears in the lightcone. If the point is *not* in the lightcone the returned position is set to the largest possible negative number in each coordinate. If the optional `positionFound` argument is given it will be set to true or false to indicate whether or not the point was found in the lightcone volume.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

<void> replicants(<integer> output→, <double(3)> nodePosition→, <replicantAction> action→, <integer> [count]←, <logical> [isInLightcone],<integer> [instance]→, <double(3)> [position]←) Performs various actions related to replicants of nodes appearing in lightcone output, depending on the value of the action argument:

- replicantActionCount returns in count the number of replicants in which the node appears in the lightcone;
- replicantActionAny returns true in isInLightcone if the given position appears in *any* replicant in the lightcone;
- replicantActionInstance returns in position the position in the instance$^{\text{th}}$ replicant in which this position appears in the lightcone.

integer(c_size_t) replicationCount(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the number of times the given nodes appears in the lightone .

double precision solidAngle() Returns the solid angle subtended by the lightcone (in units of steradians).

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision, dimension(3) velocity(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer((c_size_t))\textgreater} instance\argin) Returns the velocity vector of a node (in units of km/s) in the lightcone coordinate system.

gravitationalLensingBaryonicModifier

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((gravitationalLensingClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

`double precision magnificationCDF(\textcolor{red}{\textless double precision\textgreater} magnification\argin,\textcolor{red}{\textless double precision\textgreater} redshift\argin,\textcol double precision\textgreater} scaleSource\argin)` Returns the cumulative probability function for magnification.

`double precision magnificationPDF(\textcolor{red}{\textless double precision\textgreater} magnification\argin,\textcolor{red}{\textless double precision\textgreater} redshift\argin,\textcol double precision\textgreater} scaleSource\argin)` Returns the differential probability function for magnification.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`<void> renormalize(<double> redshift→, <double> scaleSource→)` Renormalize the gravitational lensing magnification distribution function.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

gravitationalLensingClass

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((gravitationalLensingClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision magnificationCDF(\textcolor{red}{\textless double precision\textgreater} magnification\argin,\textcolor{red}{\textless double precision\textgreater} redshift\argin,\textcol double precision\textgreater} scaleSource\argin)` Returns the cumulative probability function for magnification.

double precision magnificationPDF(\textcolor{red}{\textless double precision\textgreater} magnification\argin,\textcolor{red}{\textless double precision\textgreater} redshift\argin,\textcol double precision\textgreater} scaleSource\argin) Returns the differential probability function for magnification.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

gravitationalLensingTakahashi2011

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

<double> convergenceDistribution(<double> convergence→) Returns the gravitational lensing convergence probability density function at the given convergence.

void deepCopy(\textcolor{red}{\textless class((gravitationalLensingClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

<void> lensingDistributionConstruct(<double> redshift→, <double> scaleSource→) Construct the gravitational lensing distribution functions for the specified redshift and source scale.

double precision magnificationCDF(\textcolor{red}{\textless double precision\textgreater} magnification\argin,\textcolor{red}{\textless double precision\textgreater} redshift\argin,\textcol double precision\textgreater} scaleSource\argin) Returns the cumulative probability function for magnification.

double precision magnificationPDF(\textcolor{red}{\textless double precision\textgreater} magnification\argin,\textcolor{red}{\textless double precision\textgreater} redshift\argin,\textcol double precision\textgreater} scaleSource\argin) Returns the differential probability function for magnification.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

haloEnvironmentClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

double precision cdf(\textcolor{red}{\textless double precision\textgreater} overdensity\argin) Return the cumulative distribution function (CDF) of the environmental overdensity for the given overdensity.

void deepCopy(\textcolor{red}{\textless class((haloEnvironmentClass))\textgreater} destination\arginout Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision environmentMass() Return the mean mass contained in the region used to defined the environmental.

double precision environmentRadius() Return the radius of the region used to defined the environmental.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

`double precision overdensityLinear(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless logical\textgreater} presentDay\argin)` Return the environmental linear overdensity for the given `node`.

`double precision overdensityLinearMaximum()` Return the maximum linear overdensity for which the environmental overdensity probability density function (PDF) is non-zero.

`double precision overdensityLinearMinimum()` Return the minimum linear overdensity for which the environmental overdensity PDF is non-zero.

`void overdensityLinearSet(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} overdensity\argin)` Set the environmental overdensity for the give node.

`double precision overdensityNonLinear(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Return the environmental non-linear overdensity for the given `node`.

`double precision pdf(\textcolor{red}{\textless double precision\textgreater} overdensity\argin)` Return the PDF of the environmental overdensity for the given overdensity.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

haloEnvironmentLogNormal

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\textcolor{red}{\textless character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`double precision cdf(\textcolor{red}{\textless double precision\textgreater} overdensity\argin)` Return the CDF of the environmental overdensity for the given overdensity.

`void deepCopy(\textcolor{red}{\textless class((haloEnvironmentClass))\textgreater} destination\arginout` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\textcolor{red}{\textless logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision environmentMass()` Return the mean mass contained in the region used to defined the environmental.

`double precision environmentRadius()` Return the radius of the region used to defined the environmental.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`double precision overdensityLinear(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless logical\textgreater} presentDay\argin)` Return the environmental linear overdensity for the given `node`.

`double precision overdensityLinearMaximum()` Return the maximum linear overdensity for which the environmental overdensity PDF is non-zero.

`double precision overdensityLinearMinimum()` Return the minimum linear overdensity for which the environmental overdensity PDF is non-zero.

`void overdensityLinearSet(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textco double precision\textgreater} overdensity\argin)` Set the environmental overdensity for the give node.

`double precision overdensityNonLinear(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Return the environmental non-linear overdensity for the given `node`.

`double precision pdf(\textcolor{red}{\textless double precision\textgreater} overdensity\argin)` Return the PDF of the environmental overdensity for the given overdensity.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

haloEnvironmentNormal

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`double precision cdf(\textcolor{red}{\textless double precision\textgreater} overdensity\argin)` Return the CDF of the environmental overdensity for the given overdensity.

void deepCopy(\textcolor{red}{\textless class((haloEnvironmentClass))\textgreater} destination\arginout Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision environmentMass() Return the mean mass contained in the region used to defined the environmental.

double precision environmentRadius() Return the radius of the region used to defined the environmental.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision overdensityLinear(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless logical\textgreater} presentDay\argin) Return the environmental linear overdensity for the given node.

double precision overdensityLinearMaximum() Return the maximum linear overdensity for which the environmental overdensity PDF is non-zero.

double precision overdensityLinearMinimum() Return the minimum linear overdensity for which the environmental overdensity PDF is non-zero.

void overdensityLinearSet(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textco double precision\textgreater} overdensity\argin) Set the environmental overdensity for the give node.

double precision overdensityNonLinear(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Return the environmental non-linear overdensity for the given node.

double precision pdf(\textcolor{red}{\textless double precision\textgreater} overdensity\argin) Return the PDF of the environmental overdensity for the given overdensity.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

haloEnvironmentUniform

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`double precision cdf(\textcolor{red}{\textless double precision\textgreater} overdensity\argin)` Return the CDF of the environmental overdensity for the given overdensity.

`void deepCopy(\textcolor{red}{\textless class((haloEnvironmentClass))\textgreater} destination\arginout` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision environmentMass()` Return the mean mass contained in the region used to defined the environmental.

`double precision environmentRadius()` Return the radius of the region used to defined the environmental.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`double precision overdensityLinear(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless logical\textgreater} presentDay\argin)` Return the environmental linear overdensity for the given `node`.

`double precision overdensityLinearMaximum()` Return the maximum linear overdensity for which the environmental overdensity PDF is non-zero.

`double precision overdensityLinearMinimum()` Return the minimum linear overdensity for which the environmental overdensity PDF is non-zero.

`void overdensityLinearSet(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcc double precision\textgreater} overdensity\argin)` Set the environmental overdensity for the give node.

`double precision overdensityNonLinear(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Return the environmental non-linear overdensity for the given `node`.

`double precision pdf(\textcolor{red}{\textless double precision\textgreater} overdensity\argin)` Return the PDF of the environmental overdensity for the given overdensity.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

haloMassFunctionBhattacharya2011

`<double> a(<double> time→, <double> mass→)` Return the parameter $\bar{a}$ in the Bhattacharya et al. [2011] halo mass function fit.

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((haloMassFunctionClass))\textgreater} destination\arginou` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision differential(\textcolor{red}{\textless double precision\textgreater} time\argin,\texto double precision\textgreater} mass\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout)` Return the differential halo mass function for `mass` [$M_\odot$] at `time` [Gyr].

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`double precision integrated(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcol double precision\textgreater} massLow\argin,\textcolor{red}{\textless double precision\textgreater} massHigh\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout)` Return the halo mass function at `time` [Gyr] integrated between `massLow` and `massHigh` [$M_\odot$].

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision massFraction(\textcolor{red}{\textless double precision\textgreater} time\argin,\texto double precision\textgreater} massLow\argin,\textcolor{red}{\textless double precision\textgreater} massHigh\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout)` Return the halo mass fraction at `time` [Gyr] integrated between `massLow` and `massHigh` [$M_\odot$].

`<double> normalization(<double> time→, <double> mass→)` Return the parameter $\bar{A}$ in the Bhattacharya et al. [2011] halo mass function fit.

`type(varying_string) objectType()` Return the type of the object.

`<double> p(<double> time→, <double> mass→)` Return the parameter $\bar{p}$ in the Bhattacharya et al. [2011] halo mass function fit.

`<double> q(<double> time→, <double> mass→)` Return the parameter $\bar{q}$ in the Bhattacharya et al. [2011] halo mass function fit.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

haloMassFunctionClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((haloMassFunctionClass))\textgreater} destination\arginou Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision differential(\textcolor{red}{\textless double precision\textgreater} time\argin,\textc double precision\textgreater} mass\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout) Return the differential halo mass function for mass $[M_\odot]$ at time [Gyr].

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

double precision integrated(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcol double precision\textgreater} massLow\argin,\textcolor{red}{\textless double precision\textgreater} massHigh\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout) Return the halo mass function at time [Gyr] integrated between massLow and massHigh $[M_\odot]$.

<logical> isDefault() Return true if this is the default object of this class.

double precision massFraction(\textcolor{red}{\textless double precision\textgreater} time\argin,\textc double precision\textgreater} massLow\argin,\textcolor{red}{\textless double precision\textgreater} massHigh\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout) Return the halo mass fraction at time [Gyr] integrated between massLow and massHigh $[M_\odot]$.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

haloMassFunctionDespali2015

<double> a(<double> time→, <double> mass→) Return the parameter a in the Sheth et al. [2001] halo mass function fit.

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((haloMassFunctionClass))\textgreater} destination\arginou Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision differential(\textcolor{red}{\textless double precision\textgreater} time\argin,\textc double precision\textgreater} mass\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout) Return the differential halo mass function for mass [$M_\odot$] at time [Gyr].

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

double precision integrated(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcol double precision\textgreater} massLow\argin,\textcolor{red}{\textless double precision\textgreater} massHigh\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout) Return the halo mass function at time [Gyr] integrated between massLow and massHigh [$M_\odot$].

<logical> isDefault() Return true if this is the default object of this class.

double precision massFraction(\textcolor{red}{\textless double precision\textgreater} time\argin,\textc double precision\textgreater} massLow\argin,\textcolor{red}{\textless double precision\textgreater} massHigh\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout) Return the halo mass fraction at time [Gyr] integrated between massLow and massHigh [$M_\odot$].

<double> normalization(<double> time→, <double> mass→) Return the parameter A in the Sheth et al. [2001] halo mass function fit.

type(varying_string) objectType() Return the type of the object.

<double> p(<double> time→, <double> mass→) Return the parameter p in the Sheth et al. [2001] halo mass function fit.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

<double> x(<double> time→, <double> mass→) Return the parameter x in the Despali et al. [2015] halo mass function fit.

haloMassFunctionEnvironmental

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((haloMassFunctionClass))\textgreater} destination\arginou Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision differential(\textcolor{red}{\textless double precision\textgreater} time\argin,\textc double precision\textgreater} mass\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout) Return the differential halo mass function for mass [$M_\odot$] at time [Gyr].

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

double precision integrated(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcol double precision\textgreater} massLow\argin,\textcolor{red}{\textless double precision\textgreater} massHigh\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout) Return the halo mass function at time [Gyr] integrated between massLow and massHigh [$M_\odot$].

<logical> isDefault() Return true if this is the default object of this class.

double precision massFraction(\textcolor{red}{\textless double precision\textgreater} time\argin,\textc double precision\textgreater} massLow\argin,\textcolor{red}{\textless double precision\textgreater} massHigh\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout) Return the halo mass fraction at time [Gyr] integrated between massLow and massHigh [$M_\odot$].

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

haloMassFunctionEnvironmentAveraged

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((haloMassFunctionClass))\textgreater} destination\arginou Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision differential(\textcolor{red}{\textless double precision\textgreater} time\argin,\textc double precision\textgreater} mass\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout) Return the differential halo mass function for mass [$M_\odot$] at time [Gyr].

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

double precision integrated(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcol double precision\textgreater} massLow\argin,\textcolor{red}{\textless double precision\textgreater} massHigh\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout) Return the halo mass function at time [Gyr] integrated between massLow and massHigh [$M_\odot$].

<logical> isDefault() Return true if this is the default object of this class.

double precision massFraction(\textcolor{red}{\textless double precision\textgreater} time\argin,\textc double precision\textgreater} massLow\argin,\textcolor{red}{\textless double precision\textgreater} massHigh\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout) Return the halo mass fraction at time [Gyr] integrated between massLow and massHigh [$M_\odot$].

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

haloMassFunctionErrorConvolved

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((haloMassFunctionClass))\textgreater} destination\arginou Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision differential(\textcolor{red}{\textless double precision\textgreater} time\argin,\textc double precision\textgreater} mass\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout) Return the differential halo mass function for mass [$M_\odot$] at time [Gyr].

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

double precision integrated(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcol double precision\textgreater} massLow\argin,\textcolor{red}{\textless double precision\textgreater} massHigh\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout) Return the halo mass function at time [Gyr] integrated between massLow and massHigh [$M_\odot$].

<logical> isDefault() Return true if this is the default object of this class.

double precision massFraction(\textcolor{red}{\textless double precision\textgreater} time\argin,\textc double precision\textgreater} massLow\argin,\textcolor{red}{\textless double precision\textgreater} massHigh\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout) Return the halo mass fraction at time [Gyr] integrated between massLow and massHigh [$M_\odot$].

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

haloMassFunctionFofBias

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((haloMassFunctionClass))\textgreater} destination\arginou Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision differential(\textcolor{red}{\textless double precision\textgreater} time\argin,\textc double precision\textgreater} mass\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout) Return the differential halo mass function for mass $[M_\odot]$ at time [Gyr].

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

double precision integrated(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcol double precision\textgreater} massLow\argin,\textcolor{red}{\textless double precision\textgreater} massHigh\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout) Return the halo mass function at time [Gyr] integrated between massLow and massHigh $[M_\odot]$.

<logical> isDefault() Return true if this is the default object of this class.

double precision massFraction(\textcolor{red}{\textless double precision\textgreater} time\argin,\textc double precision\textgreater} massLow\argin,\textcolor{red}{\textless double precision\textgreater} massHigh\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout) Return the halo mass fraction at time [Gyr] integrated between massLow and massHigh $[M_\odot]$.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

haloMassFunctionPressSchechter

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((haloMassFunctionClass))\textgreater} destination\arginou` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision differential(\textcolor{red}{\textless double precision\textgreater} time\argin,\texto double precision\textgreater} mass\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout)` Return the differential halo mass function for `mass` [$M_\odot$] at `time` [Gyr].

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`double precision integrated(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcol double precision\textgreater} massLow\argin,\textcolor{red}{\textless double precision\textgreater} massHigh\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout)` Return the halo mass function at `time` [Gyr] integrated between `massLow` and `massHigh` [$M_\odot$].

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision massFraction(\textcolor{red}{\textless double precision\textgreater} time\argin,\texto double precision\textgreater} massLow\argin,\textcolor{red}{\textless double precision\textgreater} massHigh\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout)` Return the halo mass fraction at `time` [Gyr] integrated between `massLow` and `massHigh` [$M_\odot$].

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

haloMassFunctionRodriguezPuebla2016

<double> a(<double> time→, <double> mass→) Return the parameter a in the Tinker et al. [2008] halo mass function fit.

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

<double> b(<double> time→, <double> mass→) Return the parameter b in the Tinker et al. [2008] halo mass function fit.

<double> c(<double> time→, <double> mass→) Return the parameter c in the Tinker et al. [2008] halo mass function fit.

void deepCopy(\textcolor{red}{\textless class((haloMassFunctionClass))\textgreater} destination\arginou
Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision differential(\textcolor{red}{\textless double precision\textgreater} time\argin,\textc
double precision\textgreater} mass\argin,\textcolor{red}{\textless type((treeNode))\textgreater}
node \arginout) Return the differential halo mass function for mass [$M_\odot$] at time [Gyr].

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

double precision integrated(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcol
double precision\textgreater} massLow\argin,\textcolor{red}{\textless double precision\textgreater}
massHigh\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout)
Return the halo mass function at time [Gyr] integrated between massLow and massHigh [$M_\odot$].

<logical> isDefault() Return true if this is the default object of this class.

double precision massFraction(\textcolor{red}{\textless double precision\textgreater} time\argin,\textc
double precision\textgreater} massLow\argin,\textcolor{red}{\textless double precision\textgreater}
massHigh\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout)
Return the halo mass fraction at time [Gyr] integrated between massLow and massHigh [$M_\odot$].

<double> normalization(<double> time→, <double> mass→) Return the parameter A in the Tinker et al. [2008] halo mass function fit.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

haloMassFunctionShethTormen

<double> a(<double> time→, <double> mass→) Return the parameter a in the Sheth et al. [2001] halo mass function fit.

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((haloMassFunctionClass))\textgreater} destination\arginou Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision differential(\textcolor{red}{\textless double precision\textgreater} time\argin,\textc double precision\textgreater} mass\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout) Return the differential halo mass function for mass [$M_\odot$] at time [Gyr].

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

double precision integrated(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcol double precision\textgreater} massLow\argin,\textcolor{red}{\textless double precision\textgreater} massHigh\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout) Return the halo mass function at time [Gyr] integrated between massLow and massHigh [$M_\odot$].

<logical> isDefault() Return true if this is the default object of this class.

double precision massFraction(\textcolor{red}{\textless double precision\textgreater} time\argin,\textc double precision\textgreater} massLow\argin,\textcolor{red}{\textless double precision\textgreater} massHigh\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout) Return the halo mass fraction at time [Gyr] integrated between massLow and massHigh [$M_\odot$].

<double> normalization(<double> time→, <double> mass→) Return the parameter A in the Sheth et al. [2001] halo mass function fit.

type(varying_string) objectType() Return the type of the object.

<double> p(<double> time→, <double> mass→) Return the parameter p in the Sheth et al. [2001] halo mass function fit.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

haloMassFunctionSimpleSystematic

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((haloMassFunctionClass))\textgreater} destination\arginou Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision differential(\textcolor{red}{\textless double precision\textgreater} time\argin,\textc double precision\textgreater} mass\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout) Return the differential halo mass function for mass [$M_\odot$] at time [Gyr].

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

double precision integrated(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcol double precision\textgreater} massLow\argin,\textcolor{red}{\textless double precision\textgreater} massHigh\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout) Return the halo mass function at time [Gyr] integrated between massLow and massHigh [$M_\odot$].

<logical> isDefault() Return true if this is the default object of this class.

double precision massFraction(\textcolor{red}{\textless double precision\textgreater} time\argin,\textc double precision\textgreater} massLow\argin,\textcolor{red}{\textless double precision\textgreater} massHigh\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout) Return the halo mass fraction at time [Gyr] integrated between massLow and massHigh [$M_\odot$].

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

haloMassFunctionTinker2008

<double> a(<double> time→, <double> mass→) Return the parameter a in the Tinker et al. [2008] halo mass function fit.

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

<double> b(<double> time→, <double> mass→) Return the parameter b in the Tinker et al. [2008] halo mass function fit.

<double> c(<double> time→, <double> mass→) Return the parameter c in the Tinker et al. [2008] halo mass function fit.

void deepCopy(\textcolor{red}{\textless class((haloMassFunctionClass))\textgreater} destination\arginou Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision differential(\textcolor{red}{\textless double precision\textgreater} time\argin,\textc double precision\textgreater} mass\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout) Return the differential halo mass function for mass [$M_\odot$] at time [Gyr].

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

double precision integrated(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcol double precision\textgreater} massLow\argin,\textcolor{red}{\textless double precision\textgreater} massHigh\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout) Return the halo mass function at time [Gyr] integrated between massLow and massHigh [$M_\odot$].

<logical> isDefault() Return true if this is the default object of this class.

double precision massFraction(\textcolor{red}{\textless double precision\textgreater} time\argin,\textc double precision\textgreater} massLow\argin,\textcolor{red}{\textless double precision\textgreater} massHigh\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout) Return the halo mass fraction at time [Gyr] integrated between massLow and massHigh [$M_\odot$].

<double> normalization(<double> time→, <double> mass→) Return the parameter A in the Tinker et al. [2008] halo mass function fit.

type(varying_string) objectType() Return the type of the object.

<void> parametersEvaluate(<double> time→, <double> mass→) Evaluate hyper-parameters needed for the fitting function.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

haloMassFunctionTinker2008Form

<double> a(<double> time→, <double> mass→) Return the parameter a in the Tinker et al. [2008] halo mass function fit.

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

<double> b(<double> time→, <double> mass→) Return the parameter b in the Tinker et al. [2008] halo mass function fit.

<double> c(<double> time→, <double> mass→) Return the parameter c in the Tinker et al. [2008] halo mass function fit.

void deepCopy(\textcolor{red}{\textless class((haloMassFunctionClass))\textgreater} destination\arginou Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision differential(\textcolor{red}{\textless double precision\textgreater} time\argin,\textc double precision\textgreater} mass\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout) Return the differential halo mass function for mass [$M_\odot$] at time [Gyr].

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

double precision integrated(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcol double precision\textgreater} massLow\argin,\textcolor{red}{\textless double precision\textgreater} massHigh\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout) Return the halo mass function at time [Gyr] integrated between massLow and massHigh [$M_\odot$].

<logical> isDefault() Return true if this is the default object of this class.

double precision massFraction(\textcolor{red}{\textless double precision\textgreater} time\argin,\textc
double precision\textgreater} massLow\argin,\textcolor{red}{\textless double precision\textgreater}
massHigh\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout)
Return the halo mass fraction at time [Gyr] integrated between massLow and massHigh [$M_\odot$].

<double> normalization(<double> time→, <double> mass→) Return the parameter A in the Tinker et al. [2008] halo mass function fit.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

haloMassFunctionTinker2008Generic

<double> a(<double> time→, <double> mass→) Return the parameter a in the Tinker et al. [2008] halo mass function fit.

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

<double> b(<double> time→, <double> mass→) Return the parameter b in the Tinker et al. [2008] halo mass function fit.

<double> c(<double> time→, <double> mass→) Return the parameter c in the Tinker et al. [2008] halo mass function fit.

void deepCopy(\textcolor{red}{\textless class((haloMassFunctionClass))\textgreater} destination\arginou
Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision differential(\textcolor{red}{\textless double precision\textgreater} time\argin,\textc
double precision\textgreater} mass\argin,\textcolor{red}{\textless type((treeNode))\textgreater}
node \arginout) Return the differential halo mass function for mass [$M_\odot$] at time [Gyr].

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

double precision integrated(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcol
double precision\textgreater} massLow\argin,\textcolor{red}{\textless double precision\textgreater}
massHigh\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout)
Return the halo mass function at time [Gyr] integrated between massLow and massHigh [$M_\odot$].

<logical> isDefault() Return true if this is the default object of this class.

double precision massFraction(\textcolor{red}{\textless double precision\textgreater} time\argin,\texto
double precision\textgreater} massLow\argin,\textcolor{red}{\textless double precision\textgreater}
massHigh\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout)
Return the halo mass fraction at time [Gyr] integrated between massLow and massHigh [$M_\odot$].

<double> normalization(<double> time→, <double> mass→) Return the parameter A in the Tinker et al. [2008] halo mass function fit.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

haloModelPowerSpectrumModifierClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((haloModelPowerSpectrumModifierClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

void modify(\textcolor{red}{\textless double precision\textgreater} wavenumber\argin,\textcolor{red}{\t
integer\textgreater} term\argin,\textcolor{red}{\textless double precision\textgreater}
powerSpectrum\arginout,\textcolor{red}{\textless double precision\textgreater} powerSpectrumCovaria
double precision\textgreater} mass\argin) Modify the power spectra in the halo model of clustering.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

haloModelPowerSpectrumModifierIdentity

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((haloModelPowerSpectrumModifierClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`void modify(\textcolor{red}{\textless double precision\textgreater} wavenumber\argin,\textcolor{red}{\t integer\textgreater} term\argin,\textcolor{red}{\textless double precision\textgreater} powerSpectrum\arginout,\textcolor{red}{\textless double precision\textgreater} powerSpectrumCovaria double precision\textgreater} mass\argin)` Modify the power spectra in the halo model of clustering.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

haloModelPowerSpectrumModifierTriaxiality

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((haloModelPowerSpectrumModifierClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`void modify(\textcolor{red}{\textless double precision\textgreater} wavenumber\argin,\textcolor{red}{\t integer\textgreater} term\argin,\textcolor{red}{\textless double precision\textgreater} powerSpectrum\arginout,\textcolor{red}{\textless double precision\textgreater} powerSpectrumCovaria double precision\textgreater} mass\argin)` Modify the power spectra in the halo model of clustering.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

haloSpinDistributionBett2007

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((haloSpinDistributionClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision distribution(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Return the spin distribution function, $p(\lambda)$, for the given `node`. It is assumed that `node` provides the value of the spin at which the distribution function should be evaluated.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`double precision sample(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Samples a spin parameter from the distribution for the given `node`.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

haloSpinDistributionClass

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((haloSpinDistributionClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision distribution(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Return the spin distribution function, $p(\lambda)$, for the given `node`. It is assumed that `node` provides the value of the spin at which the distribution function should be evaluated.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

double precision sample(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)
Samples a spin parameter from the distribution for the given node.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

haloSpinDistributionDeltaFunction

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((haloSpinDistributionClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision distribution(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)
Return the spin distribution function, $p(\lambda)$, for the given node. It is assumed that node provides the value of the spin at which the distribution function should be evaluated.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

double precision sample(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Samples a spin parameter from the distribution for the given node.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

haloSpinDistributionLogNormal

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((haloSpinDistributionClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision distribution(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Return the spin distribution function, $p(\lambda)$, for the given node. It is assumed that node provides the value of the spin at which the distribution function should be evaluated.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

double precision sample(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Samples a spin parameter from the distribution for the given node.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

haloSpinDistributionNbodyErrors

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((haloSpinDistributionClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision distribution(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Return the spin distribution function, $p(\lambda)$, for the given node. It is assumed that node provides the value of the spin at which the distribution function should be evaluated.

<double> distributionAveraged(<*type(treeNode)> node↔, <double> massLimit→) Return the spin distribution function averaged over all halos above the given massLimit.

<double> distributionFixedPoint(<*type(treeNode)> node↔, <double> spinMeasured→) Return the spin distribution function at a fixed point in intrinsic mass and spin.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

double precision sample(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Samples a spin parameter from the distribution for the given node.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

<void> tabulate(<double> [massRequired]→, <double> [spinRequired]→) Tabulate the spin distribution as a fuction of spin and halo mass. Ensure that the table spans the massRequired and spinRequireed if provided.

hashPerfect

<void> create(<integer(kind=kind_int8)(:)> keys→, <integer(kind=kind_int8)(:)> [values]→, <logical>[keepInverseTable]→) Create a perfect hash.

<void> destroy() Destroy a perfect hash.

<integer(kind=kind_int8)> index(<integer(kind=kind_int8)> key→) Return the index corresponding to a key in a perfect hash.

<logical> isPresent(<integer(kind=kind_int8)> key→) Test if a key is present in a perfect hash.

<integer(kind=kind_int8)> size() Return the size of a perfect hash.

<integer(kind=kind_int8)> value(<integer(kind=kind_int8)> key→) Return the value corresponding to a key in a perfect hash.

hdf5Object

<void> assertAttributeType(<integer(kind=HID_T)(:)> attributeAssertedType→, <integer> attributeAssertedRank→, [matches]←) Check the type and rank of an attribute.

<void> assertDatasetType(<integer(kind=HID_T)(:)> datasetAssertedType→, <integer> datasetAssertedRank
Check the type and rank of a dataset.

<void> close() Close an HDF5 object.

<type(hdf5Object copy() Copy an HDF5 object.

<void> createReference1D(<type(hdf5Object)> toDataset↔, <character(len=*)> referenceName→, <integer(kind=HSIZE_T)(1)> referenceStart→, <integer(kind=HSIZE_T)(1)> referenceCount→) Create a reference to a 1D dataset.

<void> createReference2D(<type(hdf5Object)> toDataset↔, <character(len=*)> referenceName→, <integer(kind=HSIZE_T)(2)> referenceStart→, <integer(kind=HSIZE_T)(2)> referenceCount→) Create a reference to a 2D dataset.

<void> createReference3D(<type(hdf5Object)> toDataset↔, <character(len=*)> referenceName→, <integer(kind=HSIZE_T)(3)> referenceStart→, <integer(kind=HSIZE_T)(3)> referenceCount→) Create a reference to a 2D dataset.

<void> createReference4D(<type(hdf5Object)> toDataset↔, <character(len=*)> referenceName→, <integer(kind=HSIZE_T)(4)> referenceStart→, <integer(kind=HSIZE_T)(4)> referenceCount→) Create a reference to a 2D dataset.

<void> createReference5D(<type(hdf5Object)> toDataset↔, <character(len=*)> referenceName→, <integer(kind=HSIZE_T)(5)> referenceStart→, <integer(kind=HSIZE_T)(5)> referenceCount→) Create a reference to a 2D dataset.

<type(varying_string)> datasetNames(:) datasets() Get a list of datasets in a group object.

<void> deepCopy(<type(hdf5Object) destination←) Create a deep copy of the object with a new HDF5 object identifier.

<void> destroy() Destroy an HDF5 object.

<void> flush() Flush an HDF5 file to disk.

`<logical> hasAttribute(<character(len=*)> [attributeName]→)` Check if an object has a named attribute.

`<logical> hasDataset(<character(len=*)> [datasetName]→)` Check if an object has a named dataset.

`<logical> hasGroup(<character(len=*)> [groupName]→)` Check if an object has a named group.

`<logical> isOpen()` Return true if an object is open.

`<logical> isReference()` Return true if a dataset is a reference.

`<type(varying_string)> name()` Returns the name of a given object.

`<integer> objectType()` Return the object type.

`<type(hdf5Object openAttribute(<character(len=*)> attributeName→, <integer> [attributeDataType]→, <integer(kind=HSIZE_T)(:)> [attributeDimensions]→, <logical> [isOverwritable]→, <integer(kind=HID T)> [useDataType]→)` Open an HDF5 attribute.

`<type(hdf5Object openDataset(<character(len=*)> datasetName→, <character(len=*)> [commentText]→, <integer> [datasetDataType]→, <integer(kind=HSIZE_T)(:)> [datasetDimensions]→, <logical> [isOverwritable]→, <logical> [appendTo]→, <integer(kind=HID_T)> [useDataType]→, <integer> [chunkSize]→, <integer> [compressionLevel]→)` Open an HDF5 dataset.

`<void> openFile(<character(len=*)> [fileName], <logical> [overWrite]→, <logical> [readOnly]→, <logical> [objectsOverwritable]→, <integer> [chunkSize]→, <integer> [compressionLevel]→)` Open an HDF5 file and return an appropriate HDF5 object.

`<type(hdf5Object openGroup(<character(len=*)> groupName→, <character(len=*)> [commentText]→, <logical> [overWrite]→, <logical> [objectsOverwritable]→, <integer> [chunkSize]→, <integer> [compressionLevel]→)` Open an HDF5 group and return an appropriate HDF5 object.

`<void> parent(<character(len=*)> objectName→, <type(hdf5Object) target↔)` Return the parent object.

`<type(varying_string)> pathTo()` Returns the path to a given object.

`<integer> rank()` Return the rank of a dataset.

`<void> readAttribute(<character(len=*)> [attributeName]→), <(integer|integer(kind=kind_int8)|double|character(len=*)|type(varying_string))[:^{0-1}*]) > attributeValue←, <logical>[allowPseu` Read an attribute from an HDF5 object.

`<void> readAttributeStatic(<character(len=*)> [attributeName]→), <(integer|integer(kind=kind_int8)|double|character(len=*)|type(varying_string))[:^{0-1}]) > attributeValue←, <logical>[allowPseud` Read an attribute from an HDF5 object into a static array.

`<void> readDataset(<character(len=*)> [datasetName]→), <(integer|integer(kind=kind_int8)|double|charac string))[:^{0-1}*]) > datasetValue←, <integer(kind=HSIZE_T)[1]> [readBegin]→, <integer(kind=HSIZE_T)[1]> [readCount]→)` Read a dataset from an HDF5 group into an allocatable array.

`<void> readDatasetStatic(<character(len=*)> [datasetName]→, <(integer|integer(kind=kind_int8)|double|character(len=*)|type(varying_string))[:^{0-1}]) > datasetValue←, <integer(kind=HSIZE_T)[1]> [readBegin]→, <integer(kind=HSIZE_T)[1]> [readCount]→)` Read a dataset from an HDF5 group into a static array.

<void> readTable(<character(len=*)> [tableName]→), <character(len=*)> [columnName]→), <(integer|integer(kind=kind_int8)|double|character(len=*))[:$^{0-1}$*]) > datasetValue←, <integer(kind=HSIZE_T)[1]> [readBegin]→, <integer(kind=HSIZE_T)[1]> [readCount]→) Read a column from an HDF5 table into an allocatable array.

<integer(kind=HSIZE_T)> size(<integer> dim→) Return the size of a dataset.

<void> writeAttribute(<(integer|integer(kind=kind_int8)|double|character(len=*)|type(varying_string))(|[:])) > attributeValue→, <character(len=*)> [attributeName]→)) Write an attribute to an HDF5 object.

<void> writeDataset(<(integer|integer(kind=kind_int8)|double|character(len=*)|type(varying_string))[:$^{0-14}$]) > datasetValue→, <character(len=*)> [datasetName]→), <character(len=*)> [commentText]→, <logical>[appendTo]→, <integer>[chunkSize]→, <integer>[compressionLevel]→, <type(hdf5Object [datasetReturned]←) Write a dataset to an HDF5 group.

history

<type(history)> add(<type(history)> + <type(history)>) Addition operator.

<void> append((<double> time→, <double(:)> append→| <type(history)> append→)) Append a history or single instant onto the end of a history.

<void> builder(<*type(node)> historyDefinition→) Build a history object from an XML definition.

<void> clone(<type(history)> historyToClone→) Clone a history object.

<void> create(<integer> historyCount→, <integer> timesCount→, <double> [timeBegin]→, <double> [timeEnd]→, <integer> [rangeType]→) Creates a history object with a specified range of times.

<void> deserialize(<double(:)> historyArray→) Deserialize a history object from an array.

<void> destroy(<type(history)> historyToClone→) Destroys a history object.

<type(history)> divide(<type(history)> / <type(history)>) Division operator.

<void> dump() Dump a history object.

<void> dumpRaw(<integer> fileHandle→) Dump a history object in binary.

<logical> exists() Returns true if the given history has been created.

<void> extend(<double(2)> [timeRange]→, <double(:)> [times]→) Extends the time range of a history to encompass the specified limits.

<void> increment(<type(history)> addHistory→) Adds two histories, possibly with different time series.

<void> interpolatedIncrement(<type(history)> addHistory→) Adds two histories, possibly with different time series, by interpolating the second onto the times of the first and adding the interpolated values.

[4]For `double` datasets, up to 5-dimensional datasets are supported.

<logical> isZero() Returns true if the history is entirely zero.

<type(history)> multiply(<type(history)> * <type(history)>) Multiplication operator.

<integer(c_size_t) > nonStaticSizeOf() Returns the size of any non-static components of the type.

<void> readRaw(<integer> fileHandle→) Read a history object in binary.

<void> reset() Resets all entries in a history to zero.

<void> serialize(<double(:)> historyArray←) Serialize a history object to an array.

<integer> serializeCount() Return a count of the number of properties in a serialized history object.

<void> setToUnity() Set all entries in a history to unity.

<type(history)> subtract(<type(history)> - <type(history)>) Subtraction operator.

<void> timeSteps(<double(:)> timeSteps←) Returns an array with the timesteps (i.e. the intervals between successive times) in the given history.

<void> trim(<double> currentTime→, <integer> [minimumPointsToRemove]→) Removes any times in a history which have become outdated.

<void> trimForward(<double> currentTime→, <type(history)> [removedHistory]←) Removes any times in a history *after* the given time. Optionally returns a history object with the removed history.

hotHaloColdModeCoreRadiiClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((hotHaloColdModeCoreRadiiClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision radius(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)
Return the core radius of the hot halo mass distribution.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

hotHaloColdModeCoreRadiiVirialFraction

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((hotHaloColdModeCoreRadiiClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision radius(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Return the core radius of the hot halo mass distribution.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

hotHaloMassDistributionBetaProfile

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((hotHaloMassDistributionClass))\textgreater} destination\arginout) Perform a deep copy of the object.

double precision density(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcol double precision\textgreater} radius\argin) Return the density of the hot halo at the given radius.

double precision densityLogSlope(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} radius\argin) Return the logarithmic slope of the density of the hot halo at the given radius.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision enclosedMass(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\te double precision\textgreater} radius\argin) Return the mass enclosed in the hot halo at the given radius.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

void initialize(<*type(treeNode)> node→) Initialize the β-profile density hot halo mass distribution for the given node.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision radialMoment(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\te double precision\textgreater} moment\argin,\textcolor{red}{\textless double precision\textgreater} radius\argin) Return the density of the hot halo at the given radius.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

double precision rotationNormalization(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the relation between specific angular momentum and rotation velocity (assuming a rotation velocity that is constant in radius) for node. Specifically, the normalization, A, returned is such that $V_{\rm rot} = AJ/M$.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

hotHaloMassDistributionClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((hotHaloMassDistributionClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

double precision density(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcol
double precision\textgreater} radius\argin) Return the density of the hot halo at the given radius.

double precision densityLogSlope(\textcolor{red}{\textless type((treeNode))\textgreater}
node\arginout,\textcolor{red}{\textless double precision\textgreater} radius\argin)
Return the logarithmic slope of the density of the hot halo at the given radius.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision enclosedMass(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\te
double precision\textgreater} radius\argin) Return the mass enclosed in the hot halo at the given radius.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision radialMoment(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\te
double precision\textgreater} moment\argin,\textcolor{red}{\textless double precision\textgreater}
radius\argin) Return the density of the hot halo at the given radius.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

double precision rotationNormalization(\textcolor{red}{\textless type((treeNode))\textgreater}
node\arginout) Returns the relation between specific angular momentum and rotation velocity (assuming a rotation velocity that is constant in radius) for node. Specifically, the normalization, A, returned is such that $V_{\rm rot} = AJ/M$.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

hotHaloMassDistributionCoreRadiusClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((hotHaloMassDistributionCoreRadiusClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision radius(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Return the core radius of the hot halo mass distribution.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

hotHaloMassDistributionCoreRadiusGrowing

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((hotHaloMassDistributionCoreRadiusClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`double precision radius(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Return the core radius of the hot halo mass distribution.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

hotHaloMassDistributionCoreRadiusVirialFraction

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((hotHaloMassDistributionCoreRadiusClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`double precision radius(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Return the core radius of the hot halo mass distribution.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

hotHaloMassDistributionEnzoHydrostatic

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((hotHaloMassDistributionClass))\textgreater} destination\arginout) Perform a deep copy of the object.

double precision density(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} radius\argin) Return the density of the hot halo at the given radius.

double precision densityLogSlope(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} radius\argin) Return the logarithmic slope of the density of the hot halo at the given radius.

<double> densityNormalization(<*type(treeNode)> node↔) Return the normalization of the density profile.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\textcolor{red}{\textless logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision enclosedMass(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} radius\argin) Return the mass enclosed in the hot halo at the given radius.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDigest) Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision radialMoment(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} moment\argin,\textcolor{red}{\textless double precision\textgreater} radius\argin) Return the density of the hot halo at the given radius.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

double precision rotationNormalization(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the relation between specific angular momentum and rotation velocity (assuming a rotation velocity that is constant in radius) for node. Specifically, the normalization, A, returned is such that $V_{\rm rot} = AJ/M$.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

hotHaloMassDistributionNull

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((hotHaloMassDistributionClass))\textgreater} destination\arginout) Perform a deep copy of the object.

double precision density(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcol double precision\textgreater} radius\argin) Return the density of the hot halo at the given radius.

double precision densityLogSlope(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} radius\argin) Return the logarithmic slope of the density of the hot halo at the given radius.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision enclosedMass(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\te double precision\textgreater} radius\argin) Return the mass enclosed in the hot halo at the given radius.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

`double precision radialMoment(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\te double precision\textgreater} moment\argin,\textcolor{red}{\textless double precision\textgreater} radius\argin)` Return the density of the hot halo at the given `radius`.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`double precision rotationNormalization(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Returns the relation between specific angular momentum and rotation velocity (assuming a rotation velocity that is constant in radius) for `node`. Specifically, the normalization, A, returned is such that $V_{\rm rot} = AJ/M$.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

hotHaloMassDistributionPatejLoeb2015

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((hotHaloMassDistributionClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`double precision density(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcol double precision\textgreater} radius\argin)` Return the density of the hot halo at the given `radius`.

`double precision densityLogSlope(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} radius\argin)` Return the logarithmic slope of the density of the hot halo at the given `radius`.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision enclosedMass(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\te double precision\textgreater} radius\argin)` Return the mass enclosed in the hot halo at the given `radius`.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision radialMoment(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\te
double precision\textgreater} moment\argin,\textcolor{red}{\textless double precision\textgreater}
radius\argin) Return the density of the hot halo at the given radius.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

double precision rotationNormalization(\textcolor{red}{\textless type((treeNode))\textgreater}
node\arginout) Returns the relation between specific angular momentum and rotation velocity (assuming a rotation velocity that is constant in radius) for node. Specifically, the normalization, A, returned is such that $V_{\rm rot} = AJ/M$.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

hotHaloMassDistributionRicotti2000

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((hotHaloMassDistributionClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

double precision density(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcol
double precision\textgreater} radius\argin) Return the density of the hot halo at the given radius.

double precision densityLogSlope(\textcolor{red}{\textless type((treeNode))\textgreater}
node\arginout,\textcolor{red}{\textless double precision\textgreater} radius\argin)
Return the logarithmic slope of the density of the hot halo at the given radius.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision enclosedMass(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\te
double precision\textgreater} radius\argin) Return the mass enclosed in the hot halo at the given radius.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`void initialize(<*type(treeNode)> node→)` Initialize the β-profile density hot halo mass distribution for the given `node`.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`double precision radialMoment(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\te double precision\textgreater} moment\argin,\textcolor{red}{\textless double precision\textgreater} radius\argin)` Return the density of the hot halo at the given `radius`.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`double precision rotationNormalization(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Returns the relation between specific angular momentum and rotation velocity (assuming a rotation velocity that is constant in radius) for `node`. Specifically, the normalization, A, returned is such that $V_{\rm rot} = AJ/M$.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

hotHaloOutflowReincorporationClass

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((hotHaloOutflowReincorporationClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`double precision rate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Return the rate at which outflowed mass is being reincorporated into the hot halo.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

hotHaloOutflowReincorporationHaloDynamicalTime

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((hotHaloOutflowReincorporationClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision rate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Return the rate at which outflowed mass is being reincorporated into the hot halo.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

hotHaloOutflowReincorporationHenriques2013

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((hotHaloOutflowReincorporationClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision rate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)
Return the rate at which outflowed mass is being reincorporated into the hot halo.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

hotHaloOutflowReincorporationVelocityMaximumScaling

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

<void> calculationReset(<type(table)> node↔) Reset memoized calculations.

void deepCopy(\textcolor{red}{\textless class((hotHaloOutflowReincorporationClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision rate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)
Return the rate at which outflowed mass is being reincorporated into the hot halo.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

hotHaloOutflowReincorporationZero

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((hotHaloOutflowReincorporationClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision rate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)
Return the rate at which outflowed mass is being reincorporated into the hot halo.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

hotHaloRamPressureForceClass

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((hotHaloRamPressureForceClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision force(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Return the ram pressure force acting on `node` (in units of $(\mathrm{km/s})^2 M_\odot/\mathrm{Mpc}^3$).

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

hotHaloRamPressureForceFont2008

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((hotHaloRamPressureForceClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision force(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Return the ram pressure force acting on `node` (in units of $(\mathrm{km/s})^2 M_\odot/\mathrm{Mpc}^3$).

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

hotHaloRamPressureForceZero

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((hotHaloRamPressureForceClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision force(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Return the ram pressure force acting on `node` (in units of $(\mathrm{km/s})^2 M_\odot/\mathrm{Mpc}^3$).

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

hotHaloRamPressureStrippingClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((hotHaloRamPressureStrippingClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision radiusStripped(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Return the radius to which node is stripped due to ram pressure from its host halo (in units of Mpc).

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

hotHaloRamPressureStrippingFont2008

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((hotHaloRamPressureStrippingClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`double precision radiusStripped(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Return the radius to which `node` is stripped due to ram pressure from its host halo (in units of Mpc).

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

hotHaloRamPressureStrippingVirialRadius

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((hotHaloRamPressureStrippingClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision radiusStripped(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Return the radius to which node is stripped due to ram pressure from its host halo (in units of Mpc).

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

hotHaloRamPressureTimescaleClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((hotHaloRamPressureTimescaleClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision timescale(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Return the ram pressure stripping timescale for node (in units of Gyr).

hotHaloRamPressureTimescaleHaloDynamicalTime

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((hotHaloRamPressureTimescaleClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision timescale(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Return the ram pressure stripping timescale for node (in units of Gyr).

hotHaloRamPressureTimescaleRamPressureAcceleration

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((hotHaloRamPressureTimescaleClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision timescale(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Return the ram pressure stripping timescale for node (in units of Gyr).

hotHaloTemperatureProfileClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((hotHaloTemperatureProfileClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision temperature(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\tex
double precision\textgreater} radius\argin) Return the temperature of the hot halo at the given radius.

double precision temperatureLogSlope(\textcolor{red}{\textless type((treeNode))\textgreater}
node\arginout,\textcolor{red}{\textless double precision\textgreater} radius\argin)
Return the logarithmic slope of the temperature of the hot halo at the given radius.

hotHaloTemperatureProfileEnzoHydrostatic

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((hotHaloTemperatureProfileClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision temperature(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\tex double precision\textgreater} radius\argin) Return the temperature of the hot halo at the given radius.

double precision temperatureLogSlope(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} radius\argin) Return the logarithmic slope of the temperature of the hot halo at the given radius.

hotHaloTemperatureProfileVirial

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((hotHaloTemperatureProfileClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

`double precision temperature(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\tex double precision\textgreater} radius\argin)` Return the temperature of the hot halo at the given `radius`.

`double precision temperatureLogSlope(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} radius\argin)` Return the logarithmic slope of the temperature of the hot halo at the given `radius`.

importerUnits

`<logical> areEqual(<type(importerUnits)> units2→)` Return true if the provided units are equal.

`<logical> areNotEqual(<type(importerUnits)> units2→)` Return true if the provided units are not equal.

`<type(importerUnits)> exponentiate(<integer>exponent→)` Raise to the given integer power.

`<type(importerUnits)> multiply(<type(importerUnits)> units2→)` Multiply by another `importerUnits` object.

initialMassFunctionBaugh2005TopHeavy

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((initialMassFunctionClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) label()` Return the label for this IMF.

`double precision massMaximum()` Return the maximum mass in the initial mass function.

`double precision massMinimum()` Return the minimum mass in the initial mass function.

`type(varying_string) objectType()` Return the type of the object.

`double precision phi(\textcolor{red}{\textless double precision\textgreater} massInitial\argin)` Return the initial mass function, $\phi(M) = \mathrm{d}N/\mathrm{d}M$, at the given mass M =`massInitial`.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

void tabulate(\textcolor{red}{\textless class((table1D))\textgreater} imfTable\arginout) Return the initial mass function, $\phi(M) = \mathrm{d}N/\mathrm{d}M$, at the given mass $M =$massInitial.

initialMassFunctionBPASS

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((initialMassFunctionClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) label() Return the label for this IMF.

double precision massMaximum() Return the maximum mass in the initial mass function.

double precision massMinimum() Return the minimum mass in the initial mass function.

type(varying_string) objectType() Return the type of the object.

double precision phi(\textcolor{red}{\textless double precision\textgreater} massInitial\argin) Return the initial mass function, $\phi(M) = \mathrm{d}N/\mathrm{d}M$, at the given mass $M =$massInitial.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`void tabulate(\textcolor{red}{\textless class((table1D))\textgreater} imfTable\arginout)` Return the initial mass function, $\phi(M) = \mathrm{d}N/\mathrm{d}M$, at the given mass $M =$`initialMass`.

initialMassFunctionChabrier2001

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((initialMassFunctionClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) label()` Return the label for this IMF.

`double precision massMaximum()` Return the maximum mass in the initial mass function.

`double precision massMinimum()` Return the minimum mass in the initial mass function.

`type(varying_string) objectType()` Return the type of the object.

`double precision phi(\textcolor{red}{\textless double precision\textgreater} massInitial\argin)` Return the initial mass function, $\phi(M) = \mathrm{d}N/\mathrm{d}M$, at the given mass $M =$`massInitial`.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`void tabulate(\textcolor{red}{\textless class((table1D))\textgreater} imfTable\arginout)` Return the initial mass function, $\phi(M) = \mathrm{d}N/\mathrm{d}M$, at the given mass $M =$`initialMass`.

initialMassFunctionClass

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((initialMassFunctionClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) label()` Return the label for this IMF.

`double precision massMaximum()` Return the maximum mass in the initial mass function.

`double precision massMinimum()` Return the minimum mass in the initial mass function.

`type(varying_string) objectType()` Return the type of the object.

`double precision phi(\textcolor{red}{\textless double precision\textgreater} massInitial\argin)` Return the initial mass function, $\phi(M) = \mathrm{d}N/\mathrm{d}M$, at the given mass $M =$`massInitial`.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`void tabulate(\textcolor{red}{\textless class((table1D))\textgreater} imfTable\arginout)` Return the initial mass function, $\phi(M) = \mathrm{d}N/\mathrm{d}M$, at the given mass $M =$`initialMass`.

initialMassFunctionKennicutt1983

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((initialMassFunctionClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) label()` Return the label for this IMF.

`double precision massMaximum()` Return the maximum mass in the initial mass function.

`double precision massMinimum()` Return the minimum mass in the initial mass function.

`type(varying_string) objectType()` Return the type of the object.

`double precision phi(\textcolor{red}{\textless double precision\textgreater} massInitial\argin)` Return the initial mass function, $\phi(M) = \mathrm{d}N/\mathrm{d}M$, at the given mass $M =$`massInitial`.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`void tabulate(\textcolor{red}{\textless class((table1D))\textgreater} imfTable\arginout)` Return the initial mass function, $\phi(M) = \mathrm{d}N/\mathrm{d}M$, at the given mass $M =$`initialMass`.

initialMassFunctionKroupa2001

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((initialMassFunctionClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) label() Return the label for this IMF.

double precision massMaximum() Return the maximum mass in the initial mass function.

double precision massMinimum() Return the minimum mass in the initial mass function.

type(varying_string) objectType() Return the type of the object.

double precision phi(\textcolor{red}{\textless double precision\textgreater} massInitial\argin)
Return the initial mass function, $\phi(M) = \mathrm{d}N/\mathrm{d}M$, at the given mass $M =$massInitial.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

void tabulate(\textcolor{red}{\textless class((table1D))\textgreater} imfTable\arginout)
Return the initial mass function, $\phi(M) = \mathrm{d}N/\mathrm{d}M$, at the given mass $M =$initialMass.

initialMassFunctionMillerScalo1979

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((initialMassFunctionClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) label() Return the label for this IMF.

double precision massMaximum() Return the maximum mass in the initial mass function.

`double precision massMinimum()` Return the minimum mass in the initial mass function.

`type(varying_string) objectType()` Return the type of the object.

`double precision phi(\textcolor{red}{\textless double precision\textgreater} massInitial\argin)` Return the initial mass function, $\phi(M) = \mathrm{d}N/\mathrm{d}M$, at the given mass $M =$`massInitial`.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`void tabulate(\textcolor{red}{\textless class((table1D))\textgreater} imfTable\arginout)` Return the initial mass function, $\phi(M) = \mathrm{d}N/\mathrm{d}M$, at the given mass $M =$`initialMass`.

initialMassFunctionPiecewisePowerLaw

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((initialMassFunctionClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) label()` Return the label for this IMF.

`double precision massMaximum()` Return the maximum mass in the initial mass function.

`double precision massMinimum()` Return the minimum mass in the initial mass function.

`type(varying_string) objectType()` Return the type of the object.

`double precision phi(\textcolor{red}{\textless double precision\textgreater} massInitial\argin)` Return the initial mass function, $\phi(M) = \mathrm{d}N/\mathrm{d}M$, at the given mass $M =$`massInitial`.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

void tabulate(\textcolor{red}{\textless class((table1D))\textgreater} imfTable\arginout) Return the initial mass function, $\phi(M) = \mathrm{d}N/\mathrm{d}M$, at the given mass $M =$initialMass.

initialMassFunctionSalpeter1955

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((initialMassFunctionClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) label() Return the label for this IMF.

double precision massMaximum() Return the maximum mass in the initial mass function.

double precision massMinimum() Return the minimum mass in the initial mass function.

type(varying_string) objectType() Return the type of the object.

double precision phi(\textcolor{red}{\textless double precision\textgreater} massInitial\argin) Return the initial mass function, $\phi(M) = \mathrm{d}N/\mathrm{d}M$, at the given mass $M =$massInitial.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`void tabulate(\textcolor{red}{\textless class((table1D))\textgreater} imfTable\arginout)` Return the initial mass function, $\phi(M) = \mathrm{d}N/\mathrm{d}M$, at the given mass $M =$`initialMass`.

initialMassFunctionScalo1986

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((initialMassFunctionClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) label()` Return the label for this IMF.

`double precision massMaximum()` Return the maximum mass in the initial mass function.

`double precision massMinimum()` Return the minimum mass in the initial mass function.

`type(varying_string) objectType()` Return the type of the object.

`double precision phi(\textcolor{red}{\textless double precision\textgreater} massInitial\argin)` Return the initial mass function, $\phi(M) = \mathrm{d}N/\mathrm{d}M$, at the given mass $M =$`massInitial`.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`void tabulate(\textcolor{red}{\textless class((table1D))\textgreater} imfTable\arginout)` Return the initial mass function, $\phi(M) = \mathrm{d}N/\mathrm{d}M$, at the given mass $M =$`initialMass`.

inputParameter

<void> destroy() Destroy this parameter and all subparameters.

<void> get() Return the value of this parameter in a simple textual context.

<logical> isParameter() Return true if this is a valid parameter node.

<logical> objectCreated() Return true the object corresponding to this parameter has been created.

<*class(functionClass)> objectGet() Return a pointer to the object corresponding to this parameter.

<void> objectSet(<*class(functionClass)> object→, <logical> isShared→) Set a pointer to the object corresponding to this parameter.

<void> reset() Reset all objects in this parameter and any sub-parameters.

<void> set(<double> value→) Set the value of this parameter.

inputParameterList

<void> add(<character(len=*)> name→, <character(len=*)> value→) Add a parameter and value to the list.

<void> serializeToXML() Serialize a list of input parameters to an XML document.

inputParameters

<void> addParameter(<character(len=*)> parameterName→, <character(len=*)> parameterValue→) Add a parameter.

<void> buildTree(<type(inputParameter)> *parentParameter↔, <type(node)> *parametersNode→) Build a tree of inputParameter objects from the structure of an XML parameter file.

<void> checkParameters(<type(varying_string)[:]> [allowedParameterNames]→) Check that parameters are valid and, optionally, check if they match expected names in the provided list.

<integer> copiesCount(<character(len=*)> parameterName→, <logical>[zeroIfNotPresent]→) Return a count of the number copies of the named parameter. If the parameter is not present, this function aborts, unless zeroIfNotPresent is set to true, in which case a result of 0 is returned.

<integer> count(<character(len=*)> parameterName→, <logical>[zeroIfNotPresent]→) Return a count of the number of values in the named parameter. If the parameter is not present, this function aborts, unless zeroIfNotPresent is set to true, in which case a result of 0 is returned.

<void> destroy() Destroy the parameters document.

<logical> isGlobal() Return true if these parameters are the global set.

<logical> isPresent(<character(len=*)> parameterName→) Return true if the named parameter is present in the set.

<void> markGlobal() Mark an input parameter set as the global set.

<type(node)> node(<character(len=*)> parameterName→) Return the XML node containing the named parameter.

<void> parametersGroupCopy(<class(inputParameters)> inputParameters_→) Copy the HDF5 output group for parameters from another parameters object.

<void> parametersGroupOpen(<type(hdf5Object)> outputGroup↔) Open an output group for parameters in the given HDF5 object.

<void> reset() Reset all objects in this parameter set.

<void> resolveReferences() Resolve references in the tree of inputParameter objects.

<type(varying_string)> serializeToString() Serialize input parameters to a string.

<void> serializeToXML(<type(varying_string)> parameterFile→) Serialize input parameters to an XML file.

<type(inputParameters)> subParameters(<character(len=*)> parameterName→) Return the set of subparameters of the named parameter.

<void> validateName(<character(len=*)> parameterName→) Check that a given parameter name is a valid name, aborting if not.

<void> value((<*> parameterName|<node> parameterNode)→, parameterValue, (<*> [defaultValue])→, <inputParameterErrorStatus>[errorStatus]←, <logical>[writeOutput]→) Return the value of a parameter specified by name or XML node. A default value can be specified only if the parameter is specified by name. Supported types include rank-0 and rank-1 logicals, integers, long integers, doubles, characters, and varying strings.

integerScalarHash

<void> delete(<(character(len=*)|varying_string)> key→, <integer> value→) Delete a key from the hash.

void destroy() Destroy the hash.

<logical> exists(<(character(len=*)|varying_string|<integer>)> key→) Return true if the specified key exists in the hash.

<void> initialize() Initialize the hash.

<type(varying_string)> key(<integer> indexValue→) Return the key of the indexValue$^{\text{th}}$ entry in the hash.

<void> keys(<type(varying_string)[:]> keys↔) Return an array of all keys in the hash.

<void> set(<(character(len=*)|varying_string)> key→, <integer> value→) Set the value of a key in the hash.

<integer> size() Return the number of keys in the hash.

<integer> value(<(character(len=*)|varying_string|<integer>)> key→, <integer> value→) Return the value for the given key.

<void> values(<integer[:]> values↔) Return an array of all values in the hash.

integerSizeTScalarHash

<void> delete(<(character(len=*)|varying_string)> key→, <integer> value→) Delete a key from the hash.

void destroy() Destroy the hash.

<logical> exists(<(character(len=*)|varying_string|<integer>)> key→) Return true if the specified key exists in the hash.

<void> initialize() Initialize the hash.

<type(varying_string)> key(<integer> indexValue→) Return the key of the indexValue$^{\text{th}}$ entry in the hash.

<void> keys(<type(varying_string)[:]> keys↔) Return an array of all keys in the hash.

<void> set(<(character(len=*)|varying_string)> key→, <integer> value→) Set the value of a key in the hash.

<integer> size() Return the number of keys in the hash.

<integer> value(<(character(len=*)|varying_string|<integer>)> key→, <integer> value→) Return the value for the given key.

<void> values(<integer(c_size_t)[:]> values↔) Return an array of all values in the hash.

integrator

<void> toleranceSet(<double> [toleranceAbsolute]→, <double> [toleranceRelative]→) Set tolerances to use in this integrator.

integrator1D

<double> evaluate(<double> a→, <double> b,→) Evaluate the integral.

<void> integrandSet(<procedure(<double> (<double> x→))> integrand→) Set the integrand function to be integrated.

<void> toleranceSet(<double> [toleranceAbsolute]→, <double> [toleranceRelative]→) Set tolerances to use in this integrator.

integratorAdaptiveCompositeTrapezoidal1D

<double> evaluate(<double> a→, <double> b,→) Evaluate the integral.

<void> initialize(<integer> iterationsMaximum→) Set the maximum number of iterations allowed in the integrator.

<void> integrandSet(<procedure(<double> (<double> x→))> integrand→) Set the integrand function to be integrated.

<void> toleranceSet(<double> [toleranceAbsolute]→, <double> [toleranceRelative]→) Set tolerances to use in this integrator.

integratorCompositeGaussKronrod1D

<double> evaluate(<double> a→, <double> b,→) Evaluate the integral.

<void> evaluateInterval(<double> a→, <double> b→, <double> integralKronrod←, <double> error←) Evaluate the integral over an interval and also return the error on the integral.

<void> initialize(<integer> iterationsMaximum→, <integer> order→) Initialize the integrator.

<void> integrandSet(<procedure(<double> (<double> x→))> integrand→) Set the integrand function to be integrated.

<void> toleranceSet(<double> [toleranceAbsolute]→, <double> [toleranceRelative]→) Set tolerances to use in this integrator.

integratorCompositeTrapezoidal1D

<double> evaluate(<double> a→, <double> b,→) Evaluate the integral.

<void> initialize(<integer> iterationsMaximum→) Set the maximum number of iterations allowed in the integrator.

<void> integrandSet(<procedure(<double> (<double> x→))> integrand→) Set the integrand function to be integrated.

<void> toleranceSet(<double> [toleranceAbsolute]→, <double> [toleranceRelative]→) Set tolerances to use in this integrator.

integratorMulti

<void> tolerancesSet(<double(:)> [toleranceAbsolute]→, <double(:)> [toleranceRelative]→) Set tolerances to use in this integrator.

integratorMulti1D

<double(:)> evaluate(<double> a→, <double> b,→) Evaluate the integral.

<void> integrandSet(<integer> integrandCount→, <procedure(<double> (<double> x→))> integrand→) Set the integrand function to be integrated.

<void> tolerancesSet(<double(:)> [toleranceAbsolute]→, <double(:)> [toleranceRelative]→) Set tolerances to use in this integrator.

integratorMultiVectorized1D

<void> integrandSet(<integer> integrandCount→, <<double(:)> function(doubleone x→)> integrand→) Set the integrand function to be integrated.

<void> tolerancesSet(<double(:)> [toleranceAbsolute]→, <double(:)> [toleranceRelative]→) Set tolerances to use in this integrator.

integratorMultiVectorizedCompositeGaussKronrod1D

<void> evaluate(<double> a→, <double> b→, <double(:)> integral←) Evaluate the integrals.

<void> evaluateInterval(<double> a→, <double> b→, <double(:)> integralKronrod←, <double(:)> error←) Evaluate the integrals over an interval and also return errors on the integrals.

<void> initialize(<integer> intervalsMaximum→, <integer> order→) Set the maximum number of intervals allowed, and the order of the integrator.

<void> integrandSet(<integer> integrandCount→, <<double(:)> function(doubleone x→)> integrand→) Set the integrand function to be integrated.

<void> tolerancesSet(<double(:)> [toleranceAbsolute]→, <double(:)> [toleranceRelative]→) Set tolerances to use in this integrator.

integratorMultiVectorizedCompositeTrapezoidal1D

<void> evaluate(<double> a→, <double> b→, <double(:)> integral←) Evaluate the integrals.

<void> initialize(<integer> intervalsMaximum→) Set the maximum number of intervals allowed.

<void> integrandSet(<integer> integrandCount→, <<double(:)> function(doubleone x→)> integrand→) Set the integrand function to be integrated.

<void> tolerancesSet(<double(:)> [toleranceAbsolute]→, <double(:)> [toleranceRelative]→) Set tolerances to use in this integrator.

integratorVectorized1D

<void> integrandSet(<<double(:)> function(doubleone x→)> integrand→) Set the integrand function to be integrated.

<void> toleranceSet(<double> [toleranceAbsolute]→, <double> [toleranceRelative]→) Set tolerances to use in this integrator.

integratorVectorizedCompositeGaussKronrod1D

<double> evaluate(<double> a→, <double> b,→) Evaluate the integral.

<void> evaluateInterval(<double> a→, <double> b→, <double> integralKronrod←, <double> error←) Evaluate the integral over an interval and also return the error on the integral.

<void> initialize(<integer> iterationsMaximum→, <integer> order→) Set the maximum number of iterations allowed, and the order of the integrator.

<void> integrandSet(<<double(:)> function(doubleone x→)> integrand→) Set the integrand function to be integrated.

<void> toleranceSet(<double> [toleranceAbsolute]→, <double> [toleranceRelative]→) Set tolerances to use in this integrator.

integratorVectorizedCompositeTrapezoidal1D

<double> evaluate(<double> a→, <double> b,→) Evaluate the integral.

<void> initialize(<integer> iterationsMaximum→) Set the maximum number of iterations allowed in the integrator.

<void> integrandSet(<<double(:)> function(doubleone x→)> integrand→) Set the integrand function to be integrated.

<void> toleranceSet(<double> [toleranceAbsolute]→, <double> [toleranceRelative]→) Set tolerances to use in this integrator.

intergalacticMediumFilteringMassClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((intergalacticMediumFilteringMassClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision massFiltering(\textcolor{red}{\textless double precision\textgreater}
time\argin) Return the filtering mass at the given time.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

intergalacticMediumFilteringMassGnedin2000

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

<double(:)> coefficientsEarlyEpoch(<double> time→) Return coefficients for the early-epoch fitting function to the filtering mass.

<void> conditionsInitialODEs(<double> time→, <double(:)> massFilteringODEs←, <double(:)> [massFilteringScales]←) Set the initial conditions for the ODE system.

void deepCopy(\textcolor{red}{\textless class((intergalacticMediumFilteringMassClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision massFiltering(\textcolor{red}{\textless double precision\textgreater} time\argin) Return the filtering mass at the given time.

<double> massFilteringEarlyEpoch(<double> time→) Return the early-epoch solution for the filtering mass.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

<void> tabulate(<double> time→) Tabulate the filtering mass to encompass at least the given time.

intergalacticMediumStateClass

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((intergalacticMediumStateClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision doublyIonizedHeliumFraction(\textcolor{red}{\textless double precision\textgreater} time\argin)` Return the doubly-ionized fraction of helium in the IGM at the given time.

`double precision electronFraction(\textcolor{red}{\textless double precision\textgreater} time\argin)` Return the electron fraction (relative to hydrogen) in the IGM at the given time.

`double precision electronScatteringOpticalDepth(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless logical\textgreater} assumeFullyIonized\argin)` Return the electron scattering optical depth from the present day back to the given `time` in the IGM.

`double precision electronScatteringTime(\textcolor{red}{\textless double precision\textgreater} opticalDepth\argin,\textcolor{red}{\textless logical\textgreater} assumeFullyIonized\argin)` Return the cosmological time at which the given electron scattering `opticalDepth` is reached (integrating from the present day) in the IGM.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision massJeans(\textcolor{red}{\textless double precision\textgreater} time\argin)` Return the instantaneus Jeans mass (in $M_\odot$) at the given time.

`double precision neutralHeliumFraction(\textcolor{red}{\textless double precision\textgreater} time\argin)` Return the neutral fraction of helium in the IGM at the given time.

`double precision neutralHydrogenFraction(\textcolor{red}{\textless double precision\textgreater} time\argin)` Return the neutral fraction of hydrogen in the IGM at the given time.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`double precision singlyIonizedHeliumFraction(\textcolor{red}{\textless double precision\textgreater} time\argin)` Return the singly-ionized fraction of helium in the IGM at the given time.

double precision singlyIonizedHydrogenFraction(\textcolor{red}{\textless double precision\textgreater} time\argin) Return the singly-ionized fraction of hydrogen in the IGM at the given time.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision temperature(\textcolor{red}{\textless double precision\textgreater} time\argin) Return the temperature (in Kelvin) of the IGM at the given time.

intergalacticMediumStateFile

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((intergalacticMediumStateClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\textcolor{red}{\textless logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision doublyIonizedHeliumFraction(\textcolor{red}{\textless double precision\textgreater} time\argin) Return the doubly-ionized fraction of helium in the IGM at the given time.

double precision electronFraction(\textcolor{red}{\textless double precision\textgreater} time\argin) Return the electron fraction (relative to hydrogen) in the IGM at the given time.

double precision electronScatteringOpticalDepth(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless logical\textgreater} assumeFullyIonized\argin) Return the electron scattering optical depth from the present day back to the given time in the IGM.

double precision electronScatteringTime(\textcolor{red}{\textless double precision\textgreater} opticalDepth\argin,\textcolor{red}{\textless logical\textgreater} assumeFullyIonized\argin) Return the cosmological time at which the given electron scattering opticalDepth is reached (integrating from the present day) in the IGM.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDigest) Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision massJeans(\textcolor{red}{\textless double precision\textgreater} time\argin) Return the instantaneus Jeans mass (in $M_\odot$) at the given time.

double precision neutralHeliumFraction(\textcolor{red}{\textless double precision\textgreater} time\argin) Return the neutral fraction of helium in the IGM at the given time.

`double precision neutralHydrogenFraction(\textcolor{red}{\textless double precision\textgreater} time\argin)` Return the neutral fraction of hydrogen in the IGM at the given time.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`double precision singlyIonizedHeliumFraction(\textcolor{red}{\textless double precision\textgreater} time\argin)` Return the singly-ionized fraction of helium in the IGM at the given time.

`double precision singlyIonizedHydrogenFraction(\textcolor{red}{\textless double precision\textgreater} time\argin)` Return the singly-ionized fraction of hydrogen in the IGM at the given time.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision temperature(\textcolor{red}{\textless double precision\textgreater} time\argin)` Return the temperature (in Kelvin) of the IGM at the given time.

intergalacticMediumStateInstantReionization

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((intergalacticMediumStateClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision doublyIonizedHeliumFraction(\textcolor{red}{\textless double precision\textgreater} time\argin)` Return the doubly-ionized fraction of helium in the IGM at the given time.

`double precision electronFraction(\textcolor{red}{\textless double precision\textgreater} time\argin)` Return the electron fraction (relative to hydrogen) in the IGM at the given time.

`double precision electronScatteringOpticalDepth(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless logical\textgreater} assumeFullyIonized\argin)` Return the electron scattering optical depth from the present day back to the given `time` in the IGM.

`double precision electronScatteringTime(\textcolor{red}{\textless double precision\textgreater} opticalDepth\argin,\textcolor{red}{\textless logical\textgreater} assumeFullyIonized\argin)` Return the cosmological time at which the given electron scattering `opticalDepth` is reached (integrating from the present day) in the IGM.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision massJeans(\textcolor{red}{\textless double precision\textgreater} time\argin)` Return the instantaneus Jeans mass (in $M_\odot$) at the given time.

`double precision neutralHeliumFraction(\textcolor{red}{\textless double precision\textgreater} time\argin)` Return the neutral fraction of helium in the IGM at the given time.

`double precision neutralHydrogenFraction(\textcolor{red}{\textless double precision\textgreater} time\argin)` Return the neutral fraction of hydrogen in the IGM at the given time.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`double precision singlyIonizedHeliumFraction(\textcolor{red}{\textless double precision\textgreater} time\argin)` Return the singly-ionized fraction of helium in the IGM at the given time.

`double precision singlyIonizedHydrogenFraction(\textcolor{red}{\textless double precision\textgreater} time\argin)` Return the singly-ionized fraction of hydrogen in the IGM at the given time.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision temperature(\textcolor{red}{\textless double precision\textgreater} time\argin)` Return the temperature (in Kelvin) of the IGM at the given time.

intergalacticMediumStateInternal

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((intergalacticMediumStateClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision doublyIonizedHeliumFraction(\textcolor{red}{\textless double precision\textgreater} time\argin)` Return the doubly-ionized fraction of helium in the IGM at the given time.

`double precision electronFraction(\textcolor{red}{\textless double precision\textgreater} time\argin)` Return the electron fraction (relative to hydrogen) in the IGM at the given time.

`double precision electronScatteringOpticalDepth(\textcolor{red}{\textless double precision\textgreater time\argin,\textcolor{red}{\textless logical\textgreater} assumeFullyIonized\argin)` Return the electron scattering optical depth from the present day back to the given `time` in the IGM.

`double precision electronScatteringTime(\textcolor{red}{\textless double precision\textgreater} opticalDepth\argin,\textcolor{red}{\textless logical\textgreater} assumeFullyIonized\argin)` Return the cosmological time at which the given electron scattering `opticalDepth` is reached (integrating from the present day) in the IGM.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision massJeans(\textcolor{red}{\textless double precision\textgreater} time\argin)` Return the instantaneus Jeans mass (in $M_\odot$) at the given time.

`double precision neutralHeliumFraction(\textcolor{red}{\textless double precision\textgreater} time\argin)` Return the neutral fraction of helium in the IGM at the given time.

`double precision neutralHydrogenFraction(\textcolor{red}{\textless double precision\textgreater} time\argin)` Return the neutral fraction of hydrogen in the IGM at the given time.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`double precision singlyIonizedHeliumFraction(\textcolor{red}{\textless double precision\textgreater} time\argin)` Return the singly-ionized fraction of helium in the IGM at the given time.

`double precision singlyIonizedHydrogenFraction(\textcolor{red}{\textless double precision\textgreater} time\argin)` Return the singly-ionized fraction of hydrogen in the IGM at the given time.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision temperature(\textcolor{red}{\textless double precision\textgreater} time\argin)` Return the temperature (in Kelvin) of the IGM at the given time.

intergalacticMediumStateRecFast

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((intergalacticMediumStateClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision doublyIonizedHeliumFraction(\textcolor{red}{\textless double precision\textgreater} time\argin)` Return the doubly-ionized fraction of helium in the IGM at the given time.

`double precision electronFraction(\textcolor{red}{\textless double precision\textgreater} time\argin)` Return the electron fraction (relative to hydrogen) in the IGM at the given time.

`double precision electronScatteringOpticalDepth(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless logical\textgreater} assumeFullyIonized\argin)` Return the electron scattering optical depth from the present day back to the given `time` in the IGM.

`double precision electronScatteringTime(\textcolor{red}{\textless double precision\textgreater} opticalDepth\argin,\textcolor{red}{\textless logical\textgreater} assumeFullyIonized\argin)` Return the cosmological time at which the given electron scattering `opticalDepth` is reached (integrating from the present day) in the IGM.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision massJeans(\textcolor{red}{\textless double precision\textgreater} time\argin)` Return the instantaneus Jeans mass (in $M_\odot$) at the given time.

`double precision neutralHeliumFraction(\textcolor{red}{\textless double precision\textgreater} time\argin)` Return the neutral fraction of helium in the IGM at the given time.

`double precision neutralHydrogenFraction(\textcolor{red}{\textless double precision\textgreater} time\argin)` Return the neutral fraction of hydrogen in the IGM at the given time.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`double precision singlyIonizedHeliumFraction(\textcolor{red}{\textless double precision\textgreater} time\argin)` Return the singly-ionized fraction of helium in the IGM at the given time.

`double precision singlyIonizedHydrogenFraction(\textcolor{red}{\textless double precision\textgreater} time\argin)` Return the singly-ionized fraction of hydrogen in the IGM at the given time.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision temperature(\textcolor{red}{\textless double precision\textgreater} time\argin)` Return the temperature (in Kelvin) of the IGM at the given time.

intergalacticMediumStateSimple

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((intergalacticMediumStateClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\textcolor{red}{\textless logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision doublyIonizedHeliumFraction(\textcolor{red}{\textless double precision\textgreater} time\argin)` Return the doubly-ionized fraction of helium in the IGM at the given time.

`double precision electronFraction(\textcolor{red}{\textless double precision\textgreater} time\argin)` Return the electron fraction (relative to hydrogen) in the IGM at the given time.

`double precision electronScatteringOpticalDepth(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless logical\textgreater} assumeFullyIonized\argin)` Return the electron scattering optical depth from the present day back to the given `time` in the IGM.

`double precision electronScatteringTime(\textcolor{red}{\textless double precision\textgreater} opticalDepth\argin,\textcolor{red}{\textless logical\textgreater} assumeFullyIonized\argin)` Return the cosmological time at which the given electron scattering `opticalDepth` is reached (integrating from the present day) in the IGM.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDigest\argin)` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision massJeans(\textcolor{red}{\textless double precision\textgreater} time\argin)` Return the instantaneus Jeans mass (in $M_\odot$) at the given time.

`double precision neutralHeliumFraction(\textcolor{red}{\textless double precision\textgreater} time\argin)` Return the neutral fraction of helium in the IGM at the given time.

`double precision neutralHydrogenFraction(\textcolor{red}{\textless double precision\textgreater} time\argin)` Return the neutral fraction of hydrogen in the IGM at the given time.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`double precision singlyIonizedHeliumFraction(\textcolor{red}{\textless double precision\textgreater} time\argin)` Return the singly-ionized fraction of helium in the IGM at the given time.

`double precision singlyIonizedHydrogenFraction(\textcolor{red}{\textless double precision\textgreater} time\argin)` Return the singly-ionized fraction of hydrogen in the IGM at the given time.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision temperature(\textcolor{red}{\textless double precision\textgreater} time\argin)` Return the temperature (in Kelvin) of the IGM at the given time.

irate

`<void> copyCosmology(<type(irate)> targetFile↔)` Copy "`Cosmology`" group from one IRATE file to another.

`<void> copySimulation(<type(irate)> targetFile↔)` Copy "`SimulationProperties`" group from one IRATE file to another.

`<void> readHalos(<integer> snapshot→, <double> [redshift]←, <double(:,:)> [center]←, <double(:,:)> [velocity]←, <double(:)> [mass]←)` Read a snapshot from the IRATE format file.

`<void> readSimulation(<double> [boxSize]←)` Read the request properties of the simulation from an IRATE format file.

`<void> writeHalos(<integer> snapshot→, <double> redshift→, <double(:,:)> [center]→, <double(:,:)> [velocity]→, <double(:)> [mass]→)` HASH(0x1efc330)

keplerOrbit

<double> angularMomentum() Returns the angular momentum of an orbit.

<void> angularMomentumSet(<double> angularMomentum→) Sets the angular momentum of an orbit.

<void> assertIsDefined() Asserts that an orbit is fully defined.

<void> builder(<*type(node)> keplerOrbitDefinition→) Build a Kepler orbit from an XML definition.

<void> destroy() Destroys an orbit.

<void> dump() Dump an orbit.

<void> dumpRaw(<integer> fileHandle→) Dump an orbit in binary.

<double> eccentricity() Returns the eccentricity of an orbit.

<void> eccentricitySet(<double> eccentricity→) Sets the eccentricity of an orbit.

<double> energy() Returns the energy of an orbit.

<void> energySet(<double> energy→) Sets the energy of an orbit.

<double> epsilon() Returns the angle ϵ of an orbit.

<void> epsilonSet(<double> theta→) Sets the angle ϵ of an orbit.

<double> hostMass() Returns the host mass of an orbit.

<logical> isBound() Returns true if the orbit is bound.

<logical> isDefined() Returns true if an orbit is fully defined.

<void> massesSet(<double> satelliteMass→, <double> hostMass→) Sets the masses of satellite and host objects.

<integer(c_size_t) > nonStaticSizeOf() Returns the size of any non-static components of the type.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(:,:)> integerBuffer↔, <integer>doubleProperty↔, <integer> doubleBufferCount↔, <double(:,:)> doubleBuffer↔, <double> time→, <integer> instance→) Store a keplerOrbit object in the output buffers.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double> time→, <integer> instance→) Specify the count of a keplerOrbit object for output.

<void> outputNames(<integer> integerProperty↔, <char[*](:)> integerPropertyNames↔, <char[*](:)> integerPropertyComments↔, <double(:)> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <char[*](:)> doublePropertyNames↔, <char[*](:)> doublePropertyComments↔, <double(:)> doublePropertyUnitsSI↔, <double> time→, <integer> instance→) Specify the names of a keplerOrbit object properties for output.

<double> phi() Returns the angle ϕ of an orbit.

<void> phiSet(<double> theta→) Sets the angle ϕ of an orbit.

<coordinateCartesian position() Returns the position coordinates.

<void> postOutput(<double> time→) Perform post-output processing of a keplerOrbit object.

<void> propagate(<double> velocityRadial→) Propagates an orbit to a new position.

<double> radius() Returns the radius of an orbit.

<double> radiusApocenter() Returns the apocenter radius of an orbit.

<void> radiusApocenterSet(<double> radius→) Sets the apocenter radius of an orbit.

<double> radiusPericenter() Returns the pericenter radius of an orbit.

<void> radiusPericenterSet(<double> radius→) Sets the pericenter radius of an orbit.

<void> radiusSet(<double> radius→) Sets the radius of an orbit.

<void> readRaw(<integer> fileHandle→) Read an orbit in binary.

<void> reset() Resets an orbit to a null state.

<double> semiMajorAxis() Returns the semi-major axis of an orbit.

<void> semiMajorAxisSet(<double> semiMajorAxis→) Sets the semi-major axis of an orbit.

<double> specificReducedMass() Returns the specific reduced mass (i.e. the reduced mass per unit satellite mass, $\mu_\mathrm{s} = M_\mathrm{host}/(M_\mathrm{satellite} + M_\mathrm{host})$) of the orbit.

<double> theta() Returns the angle θ of an orbit.

<void> thetaSet(<double> theta→) Sets the angle θ of an orbit.

<coordinateCartesian velocity() Returns the velocity coordinates.

<double> velocityRadial() Returns the radial velocity of an orbit.

<void> velocityRadialSet(<double> newRadius→) Sets the radial velocity of an orbit.

<double> velocityScale() Returns the velocity scale of an orbit.

<double> velocityTangential() Returns the tangential velocity of an orbit.

<void> velocityTangentialSet(<double> velocityTangential→) Sets the tangential velocity of an orbit.

linearGrowthClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((linearGrowthClass))\textgreater} destination\arginout) Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision logarithmicDerivativeExpansionFactor(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin,\textcolor{red}{\textless logical\textgreater} collapsing \argin,\textcolor{r integer\textgreater} component \argin,\textcolor{red}{\textless double precision\textgreater} wavenumber \argin)` Return the logarithmic derivative of linear growth factor with respect to expansion factor.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision value(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{re double precision\textgreater} expansionFactor\argin,\textcolor{red}{\textless logical\textgreater} collapsing \argin,\textcolor{red}{\textless integer\textgreater} normalize\argin,\textcolor{red}{\t integer\textgreater} component \argin,\textcolor{red}{\textless double precision\textgreater} wavenumber \argin)` Return the linear growth factor at the given time and mass.

linearGrowthSimple

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((linearGrowthClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision logarithmicDerivativeExpansionFactor(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin,\textcolor{red}{\textless logical\textgreater} collapsing \argin,\textcolor{r integer\textgreater} component \argin,\textcolor{red}{\textless double precision\textgreater} wavenumber \argin)` Return the logarithmic derivative of linear growth factor with respect to expansion factor.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void retabulate(<double> time→)` Tabulate linear growth factor.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision value(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{re double precision\textgreater} expansionFactor\argin,\textcolor{red}{\textless logical\textgreater} collapsing \argin,\textcolor{red}{\textless integer\textgreater} normalize\argin,\textcolor{red}{\t integer\textgreater} component \argin,\textcolor{red}{\textless double precision\textgreater} wavenumber \argin)` Return the linear growth factor at the given time and mass.

localGroupDB

`<void> getProperty(<character(len=*)>name→, <type(varying_string)(:)>property↔)` Return an array of values of the named property for the current selection.

`<void> select(<character(len=*)>name→, <character(len=*)>value→)` Select all galaxies in the current selection where the named property has the given value.

`<void> selectAll()` Select all galaxies in the database.

`<void> update()` Update the database.

longIntegerHistory

<void> append((<double> time→, <integer(kind_int8)(:)> append→| <type(longIntegerHistory)> append→)) Append a history or single instant onto the end of a history.

<void> builder(<*type(node)> historyDefinition→) Build a history object from an XML definition.

<void> clone(<type(history)> historyToClone→) Clone a history object.

<void> create(<integer> historyCount→, <integer> timesCount→, <double> [timeBegin]→, <double> [timeEnd]→, <integer> [rangeType]→) Creates a history object with a specified range of times.

<void> destroy(<type(history)> historyToClone→) Destroys a history object.

<void> dump() Dump a history object.

<void> dumpRaw(<integer> fileHandle→) Dump a history object in binary.

<logical> exists() Returns true if the given history has been created.

<integer(c_size_t) > nonStaticSizeOf() Returns the size of any non-static components of the type.

<void> readRaw(<integer> fileHandle→) Read a history object in binary.

<void> reset() Resets all entries in a history to zero.

<void> trim(<double> currentTime→, <integer> [minimumPointsToRemove]→) Removes any times in a history which have become outdated.

<void> trimForward(<double> currentTime→, <type(longIntegerHistory)> [removedHistory]←) Removes any times in a history *after* the given time. Optionally returns a history object with the removed history.

massDistributionBetaProfile

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((massDistributionClass))\textgreater} destination\arginou Perform a deep copy of the object.

double precision density(\textcolor{red}{\textless class((coordinate))\textgreater} coordinates\argin) Return the density of the distribution at the given coordinates.

double precision densityGradientRadial(\textcolor{red}{\textless class((coordinate))\textgreater} coordinates\argin,\textcolor{red}{\textless logical\textgreater} logarithmic\argin) Return the radial gradient of density of the distribution at the given coordinates.

double precision densityRadialMoment(\textcolor{red}{\textless double precision\textgreater} moment\argin,\textcolor{red}{\textless double precision\textgreater} radiusMinimum\argin,\textcolor double precision\textgreater} radiusMaximum\argin,\textcolor{red}{\textless logical\textgreater} isInfinite\argout) Return the radial moment of the distribution.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`logical isDimensionless()` Return true if the distribution is dimensionless.

`double precision massEnclosedBySphere(\textcolor{red}{\textless double precision\textgreater} radius\argin)` Return the mass enclosed in the distribution by a sphere of given radius.

`double precision massTotal()` Return the total mass of the distribution.

`type(varying_string) objectType()` Return the type of the object.

`double precision potential(\textcolor{red}{\textless class((coordinate))\textgreater} coordinates\argin` Return the gravitational potential of the distribution at the given coordinates.

`<double> radiusHalfMass()` Returns the radius enclosing half of the mass of the mass distribution.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`integer symmetry()` Return the symmetry of the distribution.

massDistributionClass

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((massDistributionClass))\textgreater} destination\arginou` Perform a deep copy of the object.

`double precision density(\textcolor{red}{\textless class((coordinate))\textgreater} coordinates\argin)` Return the density of the distribution at the given coordinates.

`double precision densityGradientRadial(\textcolor{red}{\textless class((coordinate))\textgreater} coordinates\argin,\textcolor{red}{\textless logical\textgreater} logarithmic\argin)` Return the radial gradient of density of the distribution at the given coordinates.

double precision densityRadialMoment(\textcolor{red}{\textless double precision\textgreater} moment\argin,\textcolor{red}{\textless double precision\textgreater} radiusMinimum\argin,\textcolor{red}{\textless double precision\textgreater} radiusMaximum\argin,\textcolor{red}{\textless logical\textgreater} isInfinite\argout) Return the radial moment of the distribution.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\textcolor{red}{\textless logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDigest\argin) Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

logical isDimensionless() Return true if the distribution is dimensionless.

double precision massEnclosedBySphere(\textcolor{red}{\textless double precision\textgreater} radius\argin) Return the mass enclosed in the distribution by a sphere of given radius.

double precision massTotal() Return the total mass of the distribution.

type(varying_string) objectType() Return the type of the object.

double precision potential(\textcolor{red}{\textless class((coordinate))\textgreater} coordinates\argin) Return the gravitational potential of the distribution at the given coordinates.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

integer symmetry() Return the symmetry of the distribution.

massDistributionCylindrical

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\textcolor{red}{\textless character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((massDistributionClass))\textgreater} destination\arginout) Perform a deep copy of the object.

double precision density(\textcolor{red}{\textless class((coordinate))\textgreater} coordinates\argin) Return the density of the distribution at the given coordinates.

`double precision densityGradientRadial(\textcolor{red}{\textless class((coordinate))\textgreater} coordinates\argin,\textcolor{red}{\textless logical\textgreater} logarithmic\argin)` Return the radial gradient of density of the distribution at the given coordinates.

`double precision densityRadialMoment(\textcolor{red}{\textless double precision\textgreater} moment\argin,\textcolor{red}{\textless double precision\textgreater} radiusMinimum\argin,\textcolor{red}{\textless double precision\textgreater} radiusMaximum\argin,\textcolor{red}{\textless logical\textgreater} isInfinite\argout)` Return the radial moment of the distribution.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\textcolor{red}{\textless logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDigest\argin)` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`logical isDimensionless()` Return true if the distribution is dimensionless.

`double precision massEnclosedBySphere(\textcolor{red}{\textless double precision\textgreater} radius\argin)` Return the mass enclosed in the distribution by a sphere of given radius.

`double precision massTotal()` Return the total mass of the distribution.

`type(varying_string) objectType()` Return the type of the object.

`double precision potential(\textcolor{red}{\textless class((coordinate))\textgreater} coordinates\argin)` Return the gravitational potential of the distribution at the given coordinates.

`<double> radiusHalfMass()` Returns the cylindrical radius enclosing half of the mass of the mass distribution.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`<double> rotationCurve(<double> radius→)` Returns the circular velocity at the given `radius`.

`<double> rotationCurveGradient(<double> radius→)` Returns the gradient of the circular velocity at the given `radius`.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`<double> surfaceDensity(<class(coordinate)> coordinates→)` Returns the surface density at the given `coordinates`.

<double> surfaceDensityRadialMoment(<double> moment→, <double> radiusMinimum→, <double> radiusMaximum→, <logical> [isInfinite]←) Returns the $n^{\rm th}$ moment of the integral of the surface density over radius, $\int_0^\infty \Sigma(\mathbf{x})|x|^n \mathrm{d}\mathbf{x}$.

integer symmetry() Return the symmetry of the distribution.

massDistributionExponentialDisk

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

<double> besselFactorPotential(<double> halfRadius→) Compute the Bessel function factor appearing in the exponential disk potential.

<double> besselFactorRotationCurve(<double> halfRadius→) Compute the Bessel function factor appearing in the exponential disk rotation curve.

<double> besselFactorRotationCurveGradient(<double> halfRadius→) Compute the Bessel function factor appearing in the exponential disk rotation curve gradient.

void deepCopy(\textcolor{red}{\textless class((massDistributionClass))\textgreater} destination\arginou Perform a deep copy of the object.

double precision density(\textcolor{red}{\textless class((coordinate))\textgreater} coordinates\argin) Return the density of the distribution at the given coordinates.

double precision densityGradientRadial(\textcolor{red}{\textless class((coordinate))\textgreater} coordinates\argin,\textcolor{red}{\textless logical\textgreater} logarithmic\argin) Return the radial gradient of density of the distribution at the given coordinates.

double precision densityRadialMoment(\textcolor{red}{\textless double precision\textgreater} moment\argin,\textcolor{red}{\textless double precision\textgreater} radiusMinimum\argin,\textcolor double precision\textgreater} radiusMaximum\argin,\textcolor{red}{\textless logical\textgreater} isInfinite\argout) Return the radial moment of the distribution.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

logical isDimensionless() Return true if the distribution is dimensionless.

double precision massEnclosedBySphere(\textcolor{red}{\textless double precision\textgreater} radius\argin) Return the mass enclosed in the distribution by a sphere of given radius.

double precision massTotal() Return the total mass of the distribution.

type(varying_string) objectType() Return the type of the object.

`double precision potential(\textcolor{red}{\textless class((coordinate))\textgreater} coordinates\argin`
Return the gravitational potential of the distribution at the given coordinates.

`<double> radiusHalfMass()` Returns the cylindrical radius enclosing half of the mass of the mass distribution.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`<double> rotationCurve(<double> radius→)` Returns the circular velocity at the given `radius`.

`<double> rotationCurveGradient(<double> radius→)` Returns the gradient of the circular velocity at the given `radius`.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl`
`type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-`
`size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles`
`type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-`
`size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`<double> surfaceDensity(<class(coordinate)> coordinates→)` Returns the surface density at the given `coordinates`.

`<double> surfaceDensityRadialMoment(<double> moment→, <double> radiusMinimum→, <double> radiusMaximum→, <logical> [isInfinite]←)` Returns the $n^{\rm th}$ moment of the integral of the surface density over radius, $\int_0^\infty \Sigma(\mathbf{x})|x|^n \mathrm{d}\mathbf{x}$.

`integer symmetry()` Return the symmetry of the distribution.

`<void> tabulate()` Tabulates the potential for an exponential disk mass distribution.

massDistributionHernquist

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\`
`character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((massDistributionClass))\textgreater} destination\arginou`
Perform a deep copy of the object.

`double precision density(\textcolor{red}{\textless class((coordinate))\textgreater} coordinates\argin)`
Return the density of the distribution at the given coordinates.

`double precision densityGradientRadial(\textcolor{red}{\textless class((coordinate))\textgreater}`
`coordinates\argin,\textcolor{red}{\textless logical\textgreater} logarithmic\argin)`
Return the radial gradient of density of the distribution at the given coordinates.

double precision densityRadialMoment(\textcolor{red}{\textless double precision\textgreater} moment\argin,\textcolor{red}{\textless double precision\textgreater} radiusMinimum\argin,\textcolor{red}{\textless double precision\textgreater} radiusMaximum\argin,\textcolor{red}{\textless logical\textgreater} isInfinite\argout) Return the radial moment of the distribution.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\textcolor{red}{\textless logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDigest\argin) Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

logical isDimensionless() Return true if the distribution is dimensionless.

double precision massEnclosedBySphere(\textcolor{red}{\textless double precision\textgreater} radius\argin) Return the mass enclosed in the distribution by a sphere of given radius.

double precision massTotal() Return the total mass of the distribution.

type(varying_string) objectType() Return the type of the object.

double precision potential(\textcolor{red}{\textless class((coordinate))\textgreater} coordinates\argin) Return the gravitational potential of the distribution at the given coordinates.

<double> radiusHalfMass() Returns the radius enclosing half of the mass of the mass distribution.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

integer symmetry() Return the symmetry of the distribution.

massDistributionMiyamotoNagai

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\textcolor{red}{\textless character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((massDistributionClass))\textgreater} destination\arginout) Perform a deep copy of the object.

double precision density(\textcolor{red}{\textless class((coordinate))\textgreater} coordinates\argin) Return the density of the distribution at the given coordinates.

double precision densityGradientRadial(\textcolor{red}{\textless class((coordinate))\textgreater} coordinates\argin,\textcolor{red}{\textless logical\textgreater} logarithmic\argin) Return the radial gradient of density of the distribution at the given coordinates.

double precision densityRadialMoment(\textcolor{red}{\textless double precision\textgreater} moment\argin,\textcolor{red}{\textless double precision\textgreater} radiusMinimum\argin,\textcolor{red}{\textless double precision\textgreater} radiusMaximum\argin,\textcolor{red}{\textless logical\textgreater} isInfinite\argout) Return the radial moment of the distribution.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\textcolor{red}{\textless logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDigest) Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

logical isDimensionless() Return true if the distribution is dimensionless.

double precision massEnclosedBySphere(\textcolor{red}{\textless double precision\textgreater} radius\argin) Return the mass enclosed in the distribution by a sphere of given radius.

<void> massEnclosedTabulate() Initialize the enclosed mass tabulation.

double precision massTotal() Return the total mass of the distribution.

type(varying_string) objectType() Return the type of the object.

double precision potential(\textcolor{red}{\textless class((coordinate))\textgreater} coordinates\argin) Return the gravitational potential of the distribution at the given coordinates.

<double> radiusHalfMass() Returns the cylindrical radius enclosing half of the mass of the mass distribution.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

<double> rotationCurve(<double> radius→) Returns the circular velocity at the given radius.

<double> rotationCurveGradient(<double> radius→) Returns the gradient of the circular velocity at the given radius.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

<double> surfaceDensity(<class(coordinate)> coordinates→) Returns the surface density at the given coordinates.

<double> surfaceDensityRadialMoment(<double> moment→, <double> radiusMinimum→, <double> radiusMaximum→, <logical> [isInfinite]←) Returns the $n^{\rm th}$ moment of the integral of the surface density over radius, $\int_0^\infty \Sigma(\mathbf{x})|x|^n \mathrm{d}\mathbf{x}$.

<void> surfaceDensityTabulate() Initialize the surface density tabulation.

integer symmetry() Return the symmetry of the distribution.

massDistributionNFW

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((massDistributionClass))\textgreater} destination\arginou Perform a deep copy of the object.

double precision density(\textcolor{red}{\textless class((coordinate))\textgreater} coordinates\argin) Return the density of the distribution at the given coordinates.

double precision densityGradientRadial(\textcolor{red}{\textless class((coordinate))\textgreater} coordinates\argin,\textcolor{red}{\textless logical\textgreater} logarithmic\argin) Return the radial gradient of density of the distribution at the given coordinates.

double precision densityRadialMoment(\textcolor{red}{\textless double precision\textgreater} moment\argin,\textcolor{red}{\textless double precision\textgreater} radiusMinimum\argin,\textcolor double precision\textgreater} radiusMaximum\argin,\textcolor{red}{\textless logical\textgreater} isInfinite\argout) Return the radial moment of the distribution.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

logical isDimensionless() Return true if the distribution is dimensionless.

double precision massEnclosedBySphere(\textcolor{red}{\textless double precision\textgreater} radius\argin) Return the mass enclosed in the distribution by a sphere of given radius.

double precision massTotal() Return the total mass of the distribution.

type(varying_string) objectType() Return the type of the object.

double precision potential(\textcolor{red}{\textless class((coordinate))\textgreater} coordinates\argin Return the gravitational potential of the distribution at the given coordinates.

<double> radiusHalfMass() Returns the radius enclosing half of the mass of the mass distribution.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

integer symmetry() Return the symmetry of the distribution.

massDistributionSersic

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((massDistributionClass))\textgreater} destination\arginou Perform a deep copy of the object.

double precision density(\textcolor{red}{\textless class((coordinate))\textgreater} coordinates\argin) Return the density of the distribution at the given coordinates.

double precision densityGradientRadial(\textcolor{red}{\textless class((coordinate))\textgreater} coordinates\argin,\textcolor{red}{\textless logical\textgreater} logarithmic\argin) Return the radial gradient of density of the distribution at the given coordinates.

double precision densityRadialMoment(\textcolor{red}{\textless double precision\textgreater} moment\argin,\textcolor{red}{\textless double precision\textgreater} radiusMinimum\argin,\textcolor double precision\textgreater} radiusMaximum\argin,\textcolor{red}{\textless logical\textgreater} isInfinite\argout) Return the radial moment of the distribution.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

logical isDimensionless() Return true if the distribution is dimensionless.

double precision massEnclosedBySphere(\textcolor{red}{\textless double precision\textgreater} radius\argin) Return the mass enclosed in the distribution by a sphere of given radius.

double precision massTotal() Return the total mass of the distribution.

`type(varying_string) objectType()` Return the type of the object.

`double precision potential(\textcolor{red}{\textless class((coordinate))\textgreater} coordinates\argin` Return the gravitational potential of the distribution at the given coordinates.

`<double> radiusHalfMass()` Returns the radius enclosing half of the mass of the mass distribution.

`<double> radiusHalfMassProjected()` Return the half mass radius of the profile in projection.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`integer symmetry()` Return the symmetry of the distribution.

`<void> tabulate(<double> [radius]→)` Tabulate the Sersic profile.

massDistributionSpherical

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((massDistributionClass))\textgreater} destination\arginou` Perform a deep copy of the object.

`double precision density(\textcolor{red}{\textless class((coordinate))\textgreater} coordinates\argin)` Return the density of the distribution at the given coordinates.

`double precision densityGradientRadial(\textcolor{red}{\textless class((coordinate))\textgreater} coordinates\argin,\textcolor{red}{\textless logical\textgreater} logarithmic\argin)` Return the radial gradient of density of the distribution at the given coordinates.

`double precision densityRadialMoment(\textcolor{red}{\textless double precision\textgreater} moment\argin,\textcolor{red}{\textless double precision\textgreater} radiusMinimum\argin,\textcolor double precision\textgreater} radiusMaximum\argin,\textcolor{red}{\textless logical\textgreater} isInfinite\argout)` Return the radial moment of the distribution.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

logical isDimensionless() Return true if the distribution is dimensionless.

double precision massEnclosedBySphere(\textcolor{red}{\textless double precision\textgreater} radius\argin) Return the mass enclosed in the distribution by a sphere of given radius.

double precision massTotal() Return the total mass of the distribution.

type(varying_string) objectType() Return the type of the object.

double precision potential(\textcolor{red}{\textless class((coordinate))\textgreater} coordinates\argin
Return the gravitational potential of the distribution at the given coordinates.

<double> radiusHalfMass() Returns the radius enclosing half of the mass of the mass distribution.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

integer symmetry() Return the symmetry of the distribution.

massFunctionIncompletenessClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

double precision completeness(\textcolor{red}{\textless double precision\textgreater} mass\argin) Return the completeness of the observational sample at the given mass.

void deepCopy(\textcolor{red}{\textless class((massFunctionIncompletenessClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

massFunctionIncompletenessComplete

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

double precision completeness(\textcolor{red}{\textless double precision\textgreater} mass\argin) Return the completeness of the observational sample at the given mass.

void deepCopy(\textcolor{red}{\textless class((massFunctionIncompletenessClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

massFunctionIncompletenessSurfaceBrightness

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`double precision completeness(\textcolor{red}{\textless double precision\textgreater} mass\argin)` Return the completeness of the observational sample at the given mass.

`void deepCopy(\textcolor{red}{\textless class((massFunctionIncompletenessClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

matrix

`<void> choleskyDecompose()` Compute the Cholesky decomposition of the matrix in place.

`<void> covarianceProduct(<type(vector)> y→)` Compute $yC^{-1}y^{\mathrm{T}}$ as appears in likelihood functions utilizing covariance matrices.

`<double> determinant()` Compute and return the determinant of the matrix.

`<void> eigenSystem(<type(matrix)> eigenVectors←, <type(vector)> eigenValues←)` Compute eigenvectors and eigenvalues of the matrix.

`<type(matrix) > invert()` Compute and return the matrix inverse.

`<type(vector)> linearSystemSolve(<type(vector) y→>)` Solve the linear system $y = A \cdot x$ where A is ourself and y is the specified vector.

<double> logarithmicDeterminant() Compute and return the logarithm of the determinant of the matrix.

<type(matrix)> makeSemiPositiveDefinite() Make a matrix semi-positive definite by setting any negative eigenvalues to zero.

<void> symmetrize() Make the matrix symmetric using $A_{ij} \rightarrow (A_{ij} + A_{ji})/2$.

<type(matrix)> transpose() Return the transpose of a matrix.

mergerMassMovementsBaugh2005

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((mergerMassMovementsClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

void get(\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout,\textcolor{red}{\textle integer\textgreater} destinationGasSatellite\argout,\textcolor{red}{\textless integer\textgreater} destinationStarsSatellite\argout,\textcolor{red}{\textless integer\textgreater} destinationGasHost\ integer\textgreater} destinationStarsHost\argout,\textcolor{red}{\textless logical\textgreater} mergerIsMajor \argout) Determine movements of mass during mergers.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerMassMovementsClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((mergerMassMovementsClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

void get(\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout,\textcolor{red}{\textle integer\textgreater} destinationGasSatellite\argout,\textcolor{red}{\textless integer\textgreater} destinationStarsSatellite\argout,\textcolor{red}{\textless integer\textgreater} destinationGasHost\ integer\textgreater} destinationStarsHost\argout,\textcolor{red}{\textless logical\textgreater} mergerIsMajor \argout) Determine movements of mass during mergers.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerMassMovementsSimple

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((mergerMassMovementsClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

void get(\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout,\textcolor{red}{\textle
integer\textgreater} destinationGasSatellite\argout,\textcolor{red}{\textless integer\textgreater}
destinationStarsSatellite\argout,\textcolor{red}{\textless integer\textgreater} destinationGasHost\
integer\textgreater} destinationStarsHost\argout,\textcolor{red}{\textless logical\textgreater}
mergerIsMajor \argout) Determine movements of mass during mergers.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerMassMovementsVerySimple

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((mergerMassMovementsClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

void get(\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout,\textcolor{red}{\textle
integer\textgreater} destinationGasSatellite\argout,\textcolor{red}{\textless integer\textgreater}
destinationStarsSatellite\argout,\textcolor{red}{\textless integer\textgreater} destinationGasHost\
integer\textgreater} destinationStarsHost\argout,\textcolor{red}{\textless logical\textgreater}
mergerIsMajor \argout) Determine movements of mass during mergers.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerProgenitorPropertiesClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((mergerProgenitorPropertiesClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

void get(\textcolor{red}{\textless type((treeNode))\textgreater} nodeSatellite\arginout,\textcolor{red} type((treeNode))\textgreater} nodeHost \arginout,\textcolor{red}{\textless double precision\textgre massSatellite\argout,\textcolor{red}{\textless double precision\textgreater} massHost \argout,\textcolor{red}{\textless double precision\textgreater} massSpheroidSatellite\argout,\textc double precision\textgreater} massSpheroidHost \argout,\textcolor{red}{\textless double precision\textgreater} massSpheroidHostPreMerger\argout,\textcolor{red}{\textless double precision\textgreater} radiusSatellite \argout,\textcolor{red}{\textless double precision\textgreat radiusHost\argout,\textcolor{red}{\textless double precision\textgreater} factorAngularMomentum \argout,\textcolor{red}{\textless double precision\textgreater} massSpheroidRemnant\argout,\textcol double precision\textgreater} massGasSpheroidRemnant\argout) alculates progenitor properties for merger calculations.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerProgenitorPropertiesCole2000

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((mergerProgenitorPropertiesClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

void get(\textcolor{red}{\textless type((treeNode))\textgreater} nodeSatellite\arginout,\textcolor{red} type((treeNode))\textgreater} nodeHost \arginout,\textcolor{red}{\textless double precision\textgre massSatellite\argout,\textcolor{red}{\textless double precision\textgreater} massHost \argout,\textcolor{red}{\textless double precision\textgreater} massSpheroidSatellite\argout,\textc double precision\textgreater} massSpheroidHost \argout,\textcolor{red}{\textless double precision\textgreater} massSpheroidHostPreMerger\argout,\textcolor{red}{\textless double precision\textgreater} radiusSatellite \argout,\textcolor{red}{\textless double precision\textgreat radiusHost\argout,\textcolor{red}{\textless double precision\textgreater} factorAngularMomentum \argout,\textcolor{red}{\textless double precision\textgreater} massSpheroidRemnant\argout,\textcol double precision\textgreater} massGasSpheroidRemnant\argout) alculates progenitor properties for merger calculations.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerProgenitorPropertiesSimple

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((mergerProgenitorPropertiesClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

void get(\textcolor{red}{\textless type((treeNode))\textgreater} nodeSatellite\arginout,\textcolor{red} type((treeNode))\textgreater} nodeHost \arginout,\textcolor{red}{\textless double precision\textgre massSatellite\argout,\textcolor{red}{\textless double precision\textgreater} massHost \argout,\textcolor{red}{\textless double precision\textgreater} massSpheroidSatellite\argout,\textc double precision\textgreater} massSpheroidHost \argout,\textcolor{red}{\textless double precision\textgreater} massSpheroidHostPreMerger\argout,\textcolor{red}{\textless double precision\textgreater} radiusSatellite \argout,\textcolor{red}{\textless double precision\textgreat radiusHost\argout,\textcolor{red}{\textless double precision\textgreater} factorAngularMomentum \argout,\textcolor{red}{\textless double precision\textgreater} massSpheroidRemnant\argout,\textcol double precision\textgreater} massGasSpheroidRemnant\argout) alculates progenitor properties for merger calculations.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerProgenitorPropertiesStandard

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((mergerProgenitorPropertiesClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`void get(\textcolor{red}{\textless type((treeNode))\textgreater} nodeSatellite\arginout,\textcolor{red} type((treeNode))\textgreater} nodeHost \arginout,\textcolor{red}{\textless double precision\textgre massSatellite\argout,\textcolor{red}{\textless double precision\textgreater} massHost \argout,\textcolor{red}{\textless double precision\textgreater} massSpheroidSatellite\argout,\textc double precision\textgreater} massSpheroidHost \argout,\textcolor{red}{\textless double precision\textgreater} massSpheroidHostPreMerger\argout,\textcolor{red}{\textless double precision\textgreater} radiusSatellite \argout,\textcolor{red}{\textless double precision\textgreat radiusHost\argout,\textcolor{red}{\textless double precision\textgreater} factorAngularMomentum \argout,\textcolor{red}{\textless double precision\textgreater} massSpheroidRemnant\argout,\textcol double precision\textgreater} massGasSpheroidRemnant\argout)` alculates progenitor properties for merger calculations.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

mergerRemnantSizeClass

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((mergerRemnantSizeClass))\textgreater} destination\arginc` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`void get(\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout,\textcolor{red}{\textle double precision\textgreater} radius\argout,\textcolor{red}{\textless double precision\textgreater} velocityCircular\argout,\textcolor{red}{\textless double precision\textgreater} angularMomentumSpec` Determine merger remnant size and related properties.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

mergerRemnantSizeCole2000

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((mergerRemnantSizeClass))\textgreater} destination\arginc` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`void get(\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout,\textcolor{red}{\textle double precision\textgreater} radius\argout,\textcolor{red}{\textless double precision\textgreater} velocityCircular\argout,\textcolor{red}{\textless double precision\textgreater} angularMomentumSpec` Determine merger remnant size and related properties.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerRemnantSizeCovington2008

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((mergerRemnantSizeClass))\textgreater} destination\argino
Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

void get(\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout,\textcolor{red}{\textle
double precision\textgreater} radius\argout,\textcolor{red}{\textless double precision\textgreater}
velocityCircular\argout,\textcolor{red}{\textless double precision\textgreater} angularMomentumSpec
Determine merger remnant size and related properties.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerRemnantSizeNull

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((mergerRemnantSizeClass))\textgreater} destination\argino Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

void get(\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout,\textcolor{red}{\textle double precision\textgreater} radius\argout,\textcolor{red}{\textless double precision\textgreater} velocityCircular\argout,\textcolor{red}{\textless double precision\textgreater} angularMomentumSpec Determine merger remnant size and related properties.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerTree

<*type(treeEvent)> createEvent() Create a treeEvent object in this tree.

<void> destroy() Destroys the merger tree, including all nodes and their components.

<double> earliestTime() Return the earliest time in a merger tree.

<double> earliestTimeEvolving() Return the earliest time in an evolving merger tree.

<*type(treeNode)> getNode(<integer(kind_int8)> nodeIndex→) Returns a pointer to the node with given index in the merger tree, or a null pointer if no such node exists.

<double> latestTime() Return the latest time in a merger tree.

<void> removeEvent(<type(treeEvent)> event→) Remove a treeEvent from this tree.

<integer(c_size_t)> sizeOf() Return the size (in bytes) of a merger tree.

mergerTreeBranchingProbabilityClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((mergerTreeBranchingProbabilityClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision fractionSubresolution(\textcolor{red}{\textless double precision\textgreater}
haloMass\argin,\textcolor{red}{\textless double precision\textgreater} deltaCritical\argin,\textcol
double precision\textgreater} massResolution\argin,\textcolor{red}{\textless type((treeNode))\textg
node\arginout) Computes the fraction of subresolution mass accreted per unit “time”

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision massBranch(\textcolor{red}{\textless double precision\textgreater} haloMass\argin,\tex
double precision\textgreater} deltaCritical\argin,\textcolor{red}{\textless double
precision\textgreater} massResolution\argin,\textcolor{red}{\textless double precision\textgreater}
probabilityFraction\argin,\textcolor{red}{\textless type((pseudoRandom))\textgreater}
randomNumberGenerator\arginout,\textcolor{red}{\textless type((treeNode))\textgreater}
node\arginout) Returns the mass of a new halo created by a branching event.

type(varying_string) objectType() Return the type of the object.

double precision probability(\textcolor{red}{\textless double precision\textgreater} haloMass\argin,\te
double precision\textgreater} deltaCritical\argin,\textcolor{red}{\textless double
precision\textgreater} massResolution\argin,\textcolor{red}{\textless type((treeNode))\textgreater}
node\arginout) Computes the probability per unit “time” that a branching event occurs.

double precision probabilityBound(\textcolor{red}{\textless double precision\textgreater} haloMass\argin,\textcolor{red}{\textless double precision\textgreater} deltaCritical\argin,\textcol double precision\textgreater} massResolution\argin,\textcolor{red}{\textless integer\textgreater} bound\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Computes a bound (upper or lower) to the probability per unit "time" that a branching event occurs.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision stepMaximum(\textcolor{red}{\textless double precision\textgreater} haloMass\argin,\te double precision\textgreater} deltaCritical\argin,\textcolor{red}{\textless double precision\textgreater} massResolution\argin) Returns the maximum step in "time" allowed by this algorithm.

mergerTreeBranchingProbabilityGnrlzdPrssSchchtr

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

<void> computeCommonFactors(<double> deltaParent→, <double> massHaloParent→) Compute common factors needed for the calculations.

void deepCopy(\textcolor{red}{\textless class((mergerTreeBranchingProbabilityClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

<void> excursionSetTest(<type(treeNode)> *node↔) Compute common factors needed for the calculations.

double precision fractionSubresolution(\textcolor{red}{\textless double precision\textgreater} haloMass\argin,\textcolor{red}{\textless double precision\textgreater} deltaCritical\argin,\textcol double precision\textgreater} massResolution\argin,\textcolor{red}{\textless type((treeNode))\textg node\arginout) Computes the fraction of subresolution mass accreted per unit "time"

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision massBranch(\textcolor{red}{\textless double precision\textgreater} haloMass\argin,\tex
double precision\textgreater} deltaCritical\argin,\textcolor{red}{\textless double
precision\textgreater} massResolution\argin,\textcolor{red}{\textless double precision\textgreater}
probabilityFraction\argin,\textcolor{red}{\textless type((pseudoRandom))\textgreater}
randomNumberGenerator\arginout,\textcolor{red}{\textless type((treeNode))\textgreater}
node\arginout) Returns the mass of a new halo created by a branching event.

type(varying_string) objectType() Return the type of the object.

double precision probability(\textcolor{red}{\textless double precision\textgreater} haloMass\argin,\te
double precision\textgreater} deltaCritical\argin,\textcolor{red}{\textless double
precision\textgreater} massResolution\argin,\textcolor{red}{\textless type((treeNode))\textgreater}
node\arginout) Computes the probability per unit "time" that a branching event occurs.

double precision probabilityBound(\textcolor{red}{\textless double precision\textgreater}
haloMass\argin,\textcolor{red}{\textless double precision\textgreater} deltaCritical\argin,\textcol
double precision\textgreater} massResolution\argin,\textcolor{red}{\textless integer\textgreater}
bound\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)
Computes a bound (upper or lower) to the probability per unit "time" that a branching event occurs.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the
new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision stepMaximum(\textcolor{red}{\textless double precision\textgreater} haloMass\argin,\te
double precision\textgreater} deltaCritical\argin,\textcolor{red}{\textless double
precision\textgreater} massResolution\argin) Returns the maximum step in "time" allowed
by this algorithm.

mergerTreeBranchingProbabilityModifierClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names al-
lowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((mergerTreeBranchingProbabilityModifierClass))\textgreate
destination\arginout) Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`double precision rateModifier(\textcolor{red}{\textless double precision\textgreater} deltaParent\argin double precision\textgreater} sigmaChild\argin,\textcolor{red}{\textless double precision\textgreat sigmaParent\argin)` Return the multiplicative modifier to the tree branch probability rate.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

mergerTreeBranchingProbabilityModifierIdentity

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((mergerTreeBranchingProbabilityModifierClass))\textgreate destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`double precision rateModifier(\textcolor{red}{\textless double precision\textgreater} deltaParent\argin double precision\textgreater} sigmaChild\argin,\textcolor{red}{\textless double precision\textgreat sigmaParent\argin)` Return the multiplicative modifier to the tree branch probability rate.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerTreeBranchingProbabilityModifierParkinson2008

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((mergerTreeBranchingProbabilityModifierClass))\textgreate destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision rateModifier(\textcolor{red}{\textless double precision\textgreater} deltaParent\argin double precision\textgreater} sigmaChild\argin,\textcolor{red}{\textless double precision\textgreat sigmaParent\argin) Return the multiplicative modifier to the tree branch probability rate.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerTreeBranchingProbabilityParkinsonColeHelly

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

<void> computeCommonFactors(<double> deltaParent→, <double> massHaloParent→) Compute common factors needed for the calculations.

void deepCopy(\textcolor{red}{\textless class((mergerTreeBranchingProbabilityClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision fractionSubresolution(\textcolor{red}{\textless double precision\textgreater} haloMass\argin,\textcolor{red}{\textless double precision\textgreater} deltaCritical\argin,\textcol double precision\textgreater} massResolution\argin,\textcolor{red}{\textless type((treeNode))\textg node\arginout) Computes the fraction of subresolution mass accreted per unit "time"

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision massBranch(\textcolor{red}{\textless double precision\textgreater} haloMass\argin,\tex double precision\textgreater} deltaCritical\argin,\textcolor{red}{\textless double precision\textgreater} massResolution\argin,\textcolor{red}{\textless double precision\textgreater} probabilityFraction\argin,\textcolor{red}{\textless type((pseudoRandom))\textgreater} randomNumberGenerator\arginout,\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the mass of a new halo created by a branching event.

type(varying_string) objectType() Return the type of the object.

double precision probability(\textcolor{red}{\textless double precision\textgreater} haloMass\argin,\te double precision\textgreater} deltaCritical\argin,\textcolor{red}{\textless double precision\textgreater} massResolution\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Computes the probability per unit "time" that a branching event occurs.

double precision probabilityBound(\textcolor{red}{\textless double precision\textgreater} haloMass\argin,\textcolor{red}{\textless double precision\textgreater} deltaCritical\argin,\textcol double precision\textgreater} massResolution\argin,\textcolor{red}{\textless integer\textgreater} bound\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Computes a bound (upper or lower) to the probability per unit "time" that a branching event occurs.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision stepMaximum(\textcolor{red}{\textless double precision\textgreater} haloMass\argin,\te double precision\textgreater} deltaCritical\argin,\textcolor{red}{\textless double precision\textgreater} massResolution\argin)` Returns the maximum step in "time" allowed by this algorithm.

mergerTreeBuilderClass

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void build(\textcolor{red}{\textless type((mergerTree))\textgreater} tree\arginout)` Builds and returns a merger tree given the root `node`.

`void deepCopy(\textcolor{red}{\textless class((mergerTreeBuilderClass))\textgreater} destination\argino` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`void timeEarliestSet(\textcolor{red}{\textless double precision\textgreater} timeEarliest\argin)` Set the earliest time for the builder to the given value.

mergerTreeBuilderCole2000

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void build(\textcolor{red}{\textless type((mergerTree))\textgreater} tree\arginout) Builds and returns a merger tree given the root node.

<void> criticalOverdensitySet(<class(criticalOverdensityClass)>criticalOverdensity_) Set the critical overdensity object.

<double> criticalOverdensityUpdate(<double> deltaCritical→, <double> massCurrent→, <double> massNew→, <type(treeNode)> nodeNew↔) Set the critical overdensity object.

void deepCopy(\textcolor{red}{\textless class((mergerTreeBuilderClass))\textgreater} destination\argino Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

<logical> shouldAbort(<type(mergerTree)> tree→) Return true if construction of the merger tree should be aborted.

<logical> shouldFollowBranch(<type(mergerTree)> tree→, <type(treeNode)> node→) Return true if the branch should be followed.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

void timeEarliestSet(\textcolor{red}{\textless double precision\textgreater} timeEarliest\argin) Set the earliest time for the builder to the given value.

mergerTreeBuildMassDistributionClass

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((mergerTreeBuildMassDistributionClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`double precision sample(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{r double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} massMinimum\argin,\textcolor{red}{\textless double precision\textgreater} massMaximum\argin)` Returns the sampling rate for merger trees of the given `mass`, per decade of halo mass.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

mergerTreeBuildMassDistributionGaussian

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((mergerTreeBuildMassDistributionClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

double precision sample(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{r
double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} massMinimum\argin,\textcolor{red}{\textless double precision\textgreater} massMaximum\argin) Returns the sampling rate for merger trees of the given mass, per decade of halo mass.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerTreeBuildMassDistributionHaloMassFunction

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((mergerTreeBuildMassDistributionClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

double precision sample(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{r
double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater}
massMinimum\argin,\textcolor{red}{\textless double precision\textgreater} massMaximum\argin)
Returns the sampling rate for merger trees of the given mass, per decade of halo mass.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerTreeBuildMassDistributionPowerLaw

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((mergerTreeBuildMassDistributionClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

double precision sample(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{r
double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater}
massMinimum\argin,\textcolor{red}{\textless double precision\textgreater} massMaximum\argin)
Returns the sampling rate for merger trees of the given mass, per decade of halo mass.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerTreeBuildMassDistributionStllrMssFnctn

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((mergerTreeBuildMassDistributionClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

double precision sample(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{r double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} massMinimum\argin,\textcolor{red}{\textless double precision\textgreater} massMaximum\argin) Returns the sampling rate for merger trees of the given mass, per decade of halo mass.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerTreeBuildMassesClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void construct(\textcolor{red}{\textless double precision\textgreater} time \argin,\textcolor{red}{\tex double precision\textgreater} mass\argout,\textcolor{red}{\textless double precision\textgreater} massMinimum\argout,\textcolor{red}{\textless double precision\textgreater} massMaximum\argout,\text double precision\textgreater} weight\argout) Returns a set of merger tree masses (and either their weights, or corresponding mass interval) to be built.

`void deepCopy(\textcolor{red}{\textless class((mergerTreeBuildMassesClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

mergerTreeBuildMassesFixedMass

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void construct(\textcolor{red}{\textless double precision\textgreater} time \argin,\textcolor{red}{\tex double precision\textgreater} mass\argout,\textcolor{red}{\textless double precision\textgreater} massMinimum\argout,\textcolor{red}{\textless double precision\textgreater} massMaximum\argout,\text double precision\textgreater} weight\argout)` Returns a set of merger tree masses (and either their weights, or corresponding mass interval) to be built.

`void deepCopy(\textcolor{red}{\textless class((mergerTreeBuildMassesClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerTreeBuildMassesRead

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void construct(\textcolor{red}{\textless double precision\textgreater} time \argin,\textcolor{red}{\tex double precision\textgreater} mass\argout,\textcolor{red}{\textless double precision\textgreater} massMinimum\argout,\textcolor{red}{\textless double precision\textgreater} massMaximum\argout,\text double precision\textgreater} weight\argout) Returns a set of merger tree masses (and either their weights, or corresponding mass interval) to be built.

void deepCopy(\textcolor{red}{\textless class((mergerTreeBuildMassesClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<void> read(<double(:)> mass←, <double(:)> weight←) Read the halo masses, and, optionally, weights, from file..

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

mergerTreeBuildMassesReadHDF5

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void construct(\textcolor{red}{\textless double precision\textgreater} time \argin,\textcolor{red}{\tex double precision\textgreater} mass\argout,\textcolor{red}{\textless double precision\textgreater} massMinimum\argout,\textcolor{red}{\textless double precision\textgreater} massMaximum\argout,\text double precision\textgreater} weight\argout)` Returns a set of merger tree masses (and either their weights, or corresponding mass interval) to be built.

`void deepCopy(\textcolor{red}{\textless class((mergerTreeBuildMassesClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<void> read(<double(:)> mass←, <double(:)> weight←)` Read the halo masses, and, optionally, weights, from file..

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

mergerTreeBuildMassesReadXML

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void construct(\textcolor{red}{\textless double precision\textgreater} time \argin,\textcolor{red}{\tex
double precision\textgreater} mass\argout,\textcolor{red}{\textless double precision\textgreater}
massMinimum\argout,\textcolor{red}{\textless double precision\textgreater} massMaximum\argout,\text
double precision\textgreater} weight\argout) Returns a set of merger tree masses (and either their weights, or corresponding mass interval) to be built.

void deepCopy(\textcolor{red}{\textless class((mergerTreeBuildMassesClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<void> read(<double(:)> mass←, <double(:)> weight←) Read the halo masses, and, optionally, weights, from file..

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerTreeBuildMassesSampledDistribution

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void construct(\textcolor{red}{\textless double precision\textgreater} time \argin,\textcolor{red}{\tex double precision\textgreater} mass\argout,\textcolor{red}{\textless double precision\textgreater} massMinimum\argout,\textcolor{red}{\textless double precision\textgreater} massMaximum\argout,\text double precision\textgreater} weight\argout) Returns a set of merger tree masses (and either their weights, or corresponding mass interval) to be built.

<void> constructor(<type(inputParameters)> parameters↔) Handles construction of the abstract parent class.

void deepCopy(\textcolor{red}{\textless class((mergerTreeBuildMassesClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

<void> sampleCMF(<double(:)> x←) Return a set of values sampleCount in the interval 0–1, corresponding to values of the cumulative mass distribution.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerTreeBuildMassesSampledDistributionPseudoRandom

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void construct(\textcolor{red}{\textless double precision\textgreater} time \argin,\textcolor{red}{\tex double precision\textgreater} mass\argout,\textcolor{red}{\textless double precision\textgreater} massMinimum\argout,\textcolor{red}{\textless double precision\textgreater} massMaximum\argout,\text double precision\textgreater} weight\argout) Returns a set of merger tree masses (and either their weights, or corresponding mass interval) to be built.

<void> constructor(<type(inputParameters)> parameters↔) Handles construction of the abstract parent class.

void deepCopy(\textcolor{red}{\textless class((mergerTreeBuildMassesClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

<void> sampleCMF(<double(:)> x←) Return a set of values sampleCount in the interval 0–1, corresponding to values of the cumulative mass distribution.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerTreeBuildMassesSampledDistributionQuasiRandom

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void construct(\textcolor{red}{\textless double precision\textgreater} time \argin,\textcolor{red}{\tex double precision\textgreater} mass\argout,\textcolor{red}{\textless double precision\textgreater} massMinimum\argout,\textcolor{red}{\textless double precision\textgreater} massMaximum\argout,\text double precision\textgreater} weight\argout) Returns a set of merger tree masses (and either their weights, or corresponding mass interval) to be built.

<void> constructor(<type(inputParameters)> parameters↔) Handles construction of the abstract parent class.

void deepCopy(\textcolor{red}{\textless class((mergerTreeBuildMassesClass))\textgreater} destination\arginout) Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`<void> sampleCMF(<double(:)> x←)` Return a set of values `sampleCount` in the interval 0–1, corresponding to values of the cumulative mass distribution.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

mergerTreeBuildMassesSampledDistributionUniform

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void construct(\textcolor{red}{\textless double precision\textgreater} time \argin,\textcolor{red}{\tex double precision\textgreater} mass\argout,\textcolor{red}{\textless double precision\textgreater} massMinimum\argout,\textcolor{red}{\textless double precision\textgreater} massMaximum\argout,\text double precision\textgreater} weight\argout)` Returns a set of merger tree masses (and either their weights, or corresponding mass interval) to be built.

`<void> constructor(<type(inputParameters)> parameters↔)` Handles construction of the abstract parent class.

`void deepCopy(\textcolor{red}{\textless class((mergerTreeBuildMassesClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

<void> sampleCMF(<double(:)> x←) Return a set of values sampleCount in the interval 0–1, corresponding to values of the cumulative mass distribution.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerTreeBuildMassesUnion

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void construct(\textcolor{red}{\textless double precision\textgreater} time \argin,\textcolor{red}{\tex double precision\textgreater} mass\argout,\textcolor{red}{\textless double precision\textgreater} massMinimum\argout,\textcolor{red}{\textless double precision\textgreater} massMaximum\argout,\text double precision\textgreater} weight\argout) Returns a set of merger tree masses (and either their weights, or corresponding mass interval) to be built.

void deepCopy(\textcolor{red}{\textless class((mergerTreeBuildMassesClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

mergerTreeConstructorBuild

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`type(mergerTree), pointer construct(\textcolor{red}{\textless integer((c_size_t))\textgreater} treeNumber\argin)` Construct the merger tree corresponding to the given `treeNumber`.

`<void> constructMasses()` Construct the set of tree masses to be built.

`void deepCopy(\textcolor{red}{\textless class((mergerTreeConstructorClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

mergerTreeConstructorClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

type(mergerTree), pointer construct(\textcolor{red}{\textless integer((c_size_t))\textgreater} treeNumber\argin) Construct the merger tree corresponding to the given treeNumber.

void deepCopy(\textcolor{red}{\textless class((mergerTreeConstructorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerTreeConstructorFullySpecified

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

type(mergerTree), pointer construct(\textcolor{red}{\textless integer((c_size_t))\textgreater} treeNumber\argin) Construct the merger tree corresponding to the given treeNumber.

void deepCopy(\textcolor{red}{\textless class((mergerTreeConstructorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerTreeConstructorRead

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

<void> assignMergers(<class(nodeData)(:)> nodes↔, <class(treeNodeList)(:)> nodeList↔)
Assign pointers to merge targets.

<void> assignNamedProperties(<class(nodeData)(:)> nodes↔, <class(treeNodeList)(:)> nodeList↔)
Assign named properties to nodes.

<void> assignScaleRadii(<class(nodeData)(:)> nodes↔, <class(treeNodeList)(:)> nodeList↔)
Assign scale radii to nodes.

<void> assignSpinParameters(<class(nodeData)(:)> nodes↔, <class(treeNodeList)(:)> nodeList↔)
Assign spin parameters to nodes.

<void> assignSplitForestEvents(<class(nodeData)> node→, <class(treeNodeList)(:)> nodeList↔)
Assign events to nodes if they jump between trees in a forest.

void autoHook() Insert any event hooks required by this object.

<void> buildDescendentPointers(<class(nodeData)(:)> nodes↔) Builds pointers from each node to its descendent node.

<void> buildIsolatedParentPointers(<class(nodeData)(:)> nodes↔, <class(treeNodeList)(:)>
nodeList↔) Create parent pointer links between isolated nodes and assign times and masses to those nodes.

<void> buildSubhaloMassHistories(<class(nodeData)(:)> nodes↔, <class(treeNodeList)(:)>
nodeList↔, <integer(kind_int8)(:)> historyIndex→, <integer(c_size_t)> historyCountMaximum→,
<double(:)> historyMass↔, <double(:)> hitoryTime↔, <double(:,:)> position↔, <double(:,:)>
velocity↔) Build and attached bound mass histories to subhalos.

type(mergerTree), pointer construct(\textcolor{red}{\textless integer((c_size_t))\textgreater} treeNumber\argin) Construct the merger tree corresponding to the given treeNumber.

<void> createNodeArray(<type(mergerTree)> tree↔, <class(nodeData)(:)> nodes→, <class(nodeData)> node→, <integer> isolatedNodeCount←, <logical(:)> childIsSubhalo↔) Create an array of standard nodes and associated structures.

<void> createNodeIndices(<class(nodeData)(:)> nodes↔) Create a sorted list of node indices with an index into the original array.

void deepCopy(\textcolor{red}{\textless class((mergerTreeConstructorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

<integer(c_size_)> descendentNodeSortIndex(<integer(kind_int8)(:)> descendentIndex→) Return the sort index of the given descendentIndex.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

<void> destroyNodeIndices() Destroy the sorted list of node indices.

<void> enforceSubhaloStatus(<class(nodeData)(:)> nodes↔) Ensure that any node which was once a subhalo remains a subhalo.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

<logical> isOnPullList(<class(nodeData)> node→) Return true if the given node is on the current "pull-from" list of nodes for split forests.

<logical> isOnPushList(<class(nodeData)> node→) Return true if the given node is on the current "push-to" list of nodes for split forests.

<logical> isSubhaloSubhaloMerger(<class(nodeData)(:)> nodes→, <class(nodeData)> node→) Returns true if node undergoes a subhalo-subhalo merger.

<integer(c_size_)> nodeLocation(<integer(kind_int8)(:)> nodeIndex→) Return the location in the original array of the given nodeIndex.

type(varying_string) objectType() Return the type of the object.

<void> phaseSpacePositionRealize(<double> time→, <double(:)> position→, <double(:)> velocity→) Modify relative positions and velocities to account for both any periodicity of the simulated volume, and for Hubble flow.

<integer(c_size_t)> pullListCount(<class(nodeData)> node→) Return the number of the given node in the "pull-from" list of nodes for split forests.

<integer(c_size_t)> pullListIndex(<class(nodeData)> node→, <integer(c_size_t)> iPull→) Return the index of the given node in the "pull-from" list of nodes for split forests.

<integer(c_size_t)> pushListIndex(<class(nodeData)> node→) Return the index of the given node in the "push-to" list of nodes for split forests.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

<void> rootNodeAffinitiesInitial(<class(nodeData)(:)> nodes↔) Find initial root node affinities for all nodes.

<void> scanForBranchJumps(<class(nodeData)(:)> nodes↔, <class(treeNodeList)(:)> nodeList↔) Search for subhalos which move between branches/trees.

<void> scanForMergers(<class(nodeData)(:)> nodes↔, <class(treeNodeList)(:)> nodeList↔, <integer(c_size_t)> historyCountMaximum←) Scan for and record mergers between nodes.

<void> scanForSubhaloPromotions(<class(nodeData)(:)> nodes↔, <class(treeNodeList)(:)> nodeList↔) Scan for cases where a subhalo stops being a subhalo and so must be promoted.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

<void> timeUntilMergingSubresolution(<class(nodeData)> lastSeenNode→, <class(nodeData)(:)> nodes↔, <class(treeNodeList)(:)> nodeList↔, <integer> iNode→, <double> timeSubhaloMerges↔) Compute the additional time until merging after a subhalo is lost from the tree (presumably due to limited resolution).

<void> timingRecord(<character(len=*)> label→) Record timing data.

<void> timingReport() Report on time taken in various steps of processing merger trees read from file.

mergerTreeConstructorSmoothAccretion

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

type(mergerTree), pointer construct(\textcolor{red}{\textless integer((c_size_t))\textgreater} treeNumber\argin) Construct the merger tree corresponding to the given treeNumber.

void deepCopy(\textcolor{red}{\textless class((mergerTreeConstructorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerTreeConstructorStateRestored

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

type(mergerTree), pointer construct(\textcolor{red}{\textless integer((c_size_t))\textgreater}
treeNumber\argin) Construct the merger tree corresponding to the given treeNumber.

void deepCopy(\textcolor{red}{\textless class((mergerTreeConstructorClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerTreeData

<void> addMetadata(<metaDataType> metadataType→, <character(len=*)> label→, <(<double>,<integer>,char
value→) Add a metadatum to the tree data structure.

<void> export(<character(len=*)> outputFileName→, <mergerTreeFormat> outputFormat→, <integer>
hdfChunkSize→, <integer> hdfCompressionLevel→, <logical> [append]→) Export the tree
data to an output file.

<void> forestCountSet(<integer> forestCount→) Set the total number of forests in the data struc-
ture.

<void> makeReferences(<logical> makeReferences→) Specify whether or not merger tree dataset
references should be made.

<void> nodeCountSet(<integer> nodeCount→) Set the total number of nodes in the data structure.

<void> particleCountSet(<integer> particleCount→) Set the total number of particles in the data
structure.

<void> readASCII(inputFile→, [commentCharacter]→, [separator]→) Read node data from an
ASCII file into the data structure.

<void> readParticlesASCII(<character(len=*)> inputFile→, <character(len=1)> [commentCharacter]→,
<character(len=*)> [separator]→) Read particle data from an ASCII file into the data struc-
ture

<void> reset() Reset the data structure.

<void> setConversionFactor(<propertyType> propertyType→, <double> conversionFactor→)
Set property type and conversion factor to adjust inconsistent units.

<void> setDummyHostId(<integer> dummyHostId→) Set host ID for self-hosting halos if host ID is
not node ID.

<void> setIncludesHubbleFlow(<logical> includesHubbleFlow→) Specify if velocities include the
Hubble flow.

<void> setIncludesSubhaloMasses(<logical> includesSubhaloMasses→) Set whether halo masses
include the masses of the subhalos.

<void> setParticleMass(<double> particleMass→) Set the mass of an N-body particle in the sim-
ulation from which the trees were derived.

<void> setParticlePropertyColumn(<propertyType> propertyType→, <integer> columnNumber→)
Set the column in an ASCII data file corresponding to a given particle property.

<void> setPositionsArePeriodic(<logical> isPeriodic→) Set if positions are periodic.

<void> setProperty(<propertyType> propertyType→, <integer(kind=kind_int8)(:)|<double(:)>>
property→) Set a node property in the data structure.

<void> setPropertyColumn(<propertyType> propertyType→, columnNumber→) Set the column in an ASCII data file corresponding to a given node property.

<void> setSelfContained(<logical> areSelfContained→) Specify if trees are self-contained (i.e. contain no cross-links to other trees).

<void> setUnits(<metaDataType> unitType→, <double> unitsInSI→, <integer> [hubbleExponent]→, <integer> [scaleFactorExponent]→, <character(len=*)> [name]→) Set the units used.

mergerTreeEvolverClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((mergerTreeEvolverClass))\textgreater} destination\argino Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

void evolve(\textcolor{red}{\textless type((mergerTree))\textgreater} tree \arginout,\textcolor{red}{\ double precision\textgreater} timeEnd \argin,\textcolor{red}{\textless logical\textgreater} treeDidEvolve\argout,\textcolor{red}{\textless logical\textgreater} suspendTree\argout,\textcolor{r logical\textgreater} deadlockReporting \argin,\textcolor{red}{\textless integer((kind_- int8))\textgreater} systemClockMaximum \argin,\textcolor{red}{\textless integer((omp_- lock_kind))\textgreater} initializationLock \arginout,\textcolor{red}{\textless integer\textgreater status \argout) Evolve a merger tree.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerTreeEvolverNonEvolving

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((mergerTreeEvolverClass))\textgreater} destination\argino
Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

void evolve(\textcolor{red}{\textless type((mergerTree))\textgreater} tree \arginout,\textcolor{red}{\
double precision\textgreater} timeEnd \argin,\textcolor{red}{\textless logical\textgreater} treeDidEvolve\argout,\textcolor{red}{\textless logical\textgreater} suspendTree\argout,\textcolor{r
logical\textgreater} deadlockReporting \argin,\textcolor{red}{\textless integer((kind_-
int8))\textgreater} systemClockMaximum \argin,\textcolor{red}{\textless integer((omp_-
lock_kind))\textgreater} initializationLock \arginout,\textcolor{red}{\textless integer\textgreater
status \argout) Evolve a merger tree.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerTreeEvolverStandard

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

<void> deadlockAddNode(<type(treeNode)> *nodeLock→, <integer(kind_int8)> treeIndex→, <type(treeNode)> *node→, <type(varying_string)> lockType→) Add a node to the deadlock list.

<void> deadlockOutputTree(<double> timeEnd→) Output a description of a deadlocked tree.

void deepCopy(\textcolor{red}{\textless class((mergerTreeEvolverClass))\textgreater} destination\arginc Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

void evolve(\textcolor{red}{\textless type((mergerTree))\textgreater} tree \arginout,\textcolor{red}{\ double precision\textgreater} timeEnd \argin,\textcolor{red}{\textless logical\textgreater} treeDidEvolve\argout,\textcolor{red}{\textless logical\textgreater} suspendTree\argout,\textcolor{r logical\textgreater} deadlockReporting \argin,\textcolor{red}{\textless integer((kind_- int8))\textgreater} systemClockMaximum \argin,\textcolor{red}{\textless integer((omp_- lock_kind))\textgreater} initializationLock \arginout,\textcolor{red}{\textless integer\textgreater status \argout) Evolve a merger tree.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

<double> timeEvolveTo(<type(treeNode)> *node↔, <double> timeEnd→, <procedure(timestepTask)> *timestepTask_←, <class(*)> *timestepSelf ←, <logical> report→, <type(treeNode)> [*nodeLock]←, <type(varying_string)> [lockType]←) Find the time to which a node can be evolved.

mergerTreeEvolveTimestepClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((mergerTreeEvolveTimestepClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision timeEvolveTo(\textcolor{red}{\textless type((treeNode))\textgreater}
node \arginout,\textcolor{red}{\textless procedure((timestepTask))\textgreater} task
\argout,\textcolor{red}{\textless class((*))\textgreater} taskSelf\argout,\textcolor{red}{\textles
logical\textgreater} report \argin,\textcolor{red}{\textless type((treeNode))\textgreater}
lockNode\argout,\textcolor{red}{\textless type((varying_string))\textgreater} lockType\argout)
Return the time to which the node can be evolved.

mergerTreeEvolveTimestepHistory

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((mergerTreeEvolveTimestepClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision timeEvolveTo(\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout,\textcolor{red}{\textless procedure((timestepTask))\textgreater} task \argout,\textcolor{red}{\textless class((*))\textgreater} taskSelf\argout,\textcolor{red}{\textles logical\textgreater} report \argin,\textcolor{red}{\textless type((treeNode))\textgreater} lockNode\argout,\textcolor{red}{\textless type((varying_string))\textgreater} lockType\argout) Return the time to which the node can be evolved.

mergerTreeEvolveTimestepMulti

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((mergerTreeEvolveTimestepClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision timeEvolveTo(\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout,\textcolor{red}{\textless procedure((timestepTask))\textgreater} task \argout,\textcolor{red}{\textless class((*))\textgreater} taskSelf\argout,\textcolor{red}{\textles logical\textgreater} report \argin,\textcolor{red}{\textless type((treeNode))\textgreater} lockNode\argout,\textcolor{red}{\textless type((varying_string))\textgreater} lockType\argout)` Return the time to which the node can be evolved.

mergerTreeEvolveTimestepRecordEvolution

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((mergerTreeEvolveTimestepClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`<void> reset()` Reset the record of galaxy evolution.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision timeEvolveTo(\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout,\textcolor{red}{\textless procedure((timestepTask))\textgreater} task \argout,\textcolor{red}{\textless class((*))\textgreater} taskSelf\argout,\textcolor{red}{\textles logical\textgreater} report \argin,\textcolor{red}{\textless type((treeNode))\textgreater} lockNode\argout,\textcolor{red}{\textless type((varying_string))\textgreater} lockType\argout)` Return the time to which the node can be evolved.

mergerTreeEvolveTimestepSatellite

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((mergerTreeEvolveTimestepClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision timeEvolveTo(\textcolor{red}{\textless type((treeNode))\textgreater}
node \arginout,\textcolor{red}{\textless procedure((timestepTask))\textgreater} task
\argout,\textcolor{red}{\textless class((*))\textgreater} taskSelf\argout,\textcolor{red}{\textles
logical\textgreater} report \argin,\textcolor{red}{\textless type((treeNode))\textgreater}
lockNode\argout,\textcolor{red}{\textless type((varying_string))\textgreater} lockType\argout)
Return the time to which the node can be evolved.

mergerTreeEvolveTimestepSimple

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((mergerTreeEvolveTimestepClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision timeEvolveTo(\textcolor{red}{\textless type((treeNode))\textgreater}
node \arginout,\textcolor{red}{\textless procedure((timestepTask))\textgreater} task
\argout,\textcolor{red}{\textless class((*))\textgreater} taskSelf\argout,\textcolor{red}{\textles
logical\textgreater} report \argin,\textcolor{red}{\textless type((treeNode))\textgreater}
lockNode\argout,\textcolor{red}{\textless type((varying_string))\textgreater} lockType\argout)
Return the time to which the node can be evolved.

mergerTreeEvolveTimestepStandard

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((mergerTreeEvolveTimestepClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision timeEvolveTo(\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout,\textcolor{red}{\textless procedure((timestepTask))\textgreater} task \argout,\textcolor{red}{\textless class((*))\textgreater} taskSelf\argout,\textcolor{red}{\textles logical\textgreater} report \argin,\textcolor{red}{\textless type((treeNode))\textgreater} lockNode\argout,\textcolor{red}{\textless type((varying_string))\textgreater} lockType\argout) Return the time to which the node can be evolved.

mergerTreeImporterClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

logical angularMomenta3DAvailable() Return true if angular momenta (vectors) are available.

logical angularMomentaAvailable() Return true if angular momenta (magnitudes) are available.

logical angularMomentaIncludeSubhalos() Returns a Boolean specifying whether halo angular momenta (or spins) include the contribution from their subhalos.

void autoHook() Insert any event hooks required by this object.

logical canReadSubsets() Returns true if arbitrary subsets of halos from a forest can be read.

void close() Closes the file.

double precision cubeLength(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcol integer\textgreater} status\argout) Returns the length of the simulation cube.

void deepCopy(\textcolor{red}{\textless class((mergerTreeImporterClass))\textgreater} destination\argin Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`void import(\textcolor{red}{\textless integer\textgreater} i\argin,\textcolor{red}{\textless class((nodeDataMinimal))\textgreater} nodes\argout,\textcolor{red}{\textless integer((c_-size_t))\textgreater} nodeSubset\argin,\textcolor{red}{\textless logical\textgreater} requireScaleRadii\argin,\textcolor{red}{\textless logical\textgreater} requireAngularMomenta\argin,logical\textgreater} requireAngularMomenta3D\argin,\textcolor{red}{\textless logical\textgreater} requireSpin\argin,\textcolor{red}{\textless logical\textgreater} requireSpin3D\argin,\textcolor{redlogical\textgreater} requirePositions\argin,\textcolor{red}{\textless logical\textgreater} structureOnly\argin,\textcolor{red}{\textless type((varying_string))\textgreater} requireNamedReals\argin,\textcolor{red}{\textless type((varying_string))\textgreater} requireNamedIntegers\argin)` Imports the i^{th} tree.

`<logical> isDefault()` Return true if this is the default object of this class.

`logical massesIncludeSubhalos()` Returns a Boolean specifying whether halo masses include the contribution from their subhalos.

`integer(kind=c_size_t) nodeCount(\textcolor{red}{\textless integer\textgreater} i\argin)` Returns the number of nodes in the i^{th} tree.

`type(varying_string) objectType()` Return the type of the object.

`void open(\textcolor{red}{\textless type((varying_string))\textgreater} fileName\argin)` Opens the file.

`logical particleCountAvailable()` Return true if particle counts are available.

`integer positionsArePeriodic()` Returns a Boolean integer specifying whether positions are periodic.

`logical positionsAvailable(\textcolor{red}{\textless logical\textgreater} positions\argin,\textcolor{relogical\textgreater} velocities\argin)` Return true if positions and/or velocities are available.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`logical scaleRadiiAvailable()` Return true if scale radii are available.

`logical spin3DAvailable()` Return true if spin (vectors) are available.

`logical spinAvailable()` Return true if spin (magnitudes) are available.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textltype((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textlestype((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`void subhaloTrace(\textcolor{red}{\textless class((nodeData))\textgreater} node\argin,\textcolor{red}{\double precision\textgreater} time\argout,\textcolor{red}{\textless double precision\textgreater} position\argout,\textcolor{red}{\textless double precision\textgreater} velocity\argout)` Supplies epochs, positions, and velocities for traced subhalos.

`integer(kind=c_size_t) subhaloTraceCount(\textcolor{red}{\textless class((nodeData))\textgreater} node\argin)` Returns the length of a node's subhalo trace.

`integer(kind=c_size_t) treeCount()` Returns a count of the number of trees available.

`integer(kind=kind_int8) treeIndex(\textcolor{red}{\textless integer\textgreater} i\argin)` Returns the index of the $i^{\rm th}$ tree.

`integer treesAreSelfContained()` Returns a Boolean integer specifying whether trees are self-contained.

`integer treesHaveSubhalos()` Returns a Boolean integer specifying whether or not the trees have subhalos.

`double precision treeWeight(\textcolor{red}{\textless integer\textgreater} i\argin)` Returns the weight to assign to the $i^{\rm th}$ tree.

`integer velocitiesIncludeHubbleFlow()` Returns a Boolean integer specifying whether velocities include the Hubble flow.

`logical velocityDispersionAvailable()` Return true if halo velocity dispersions are available.

`logical velocityMaximumAvailable()` Return true if rotation curve velocity maxima are available.

mergerTreeImporterGalacticus

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`logical angularMomenta3DAvailable()` Return true if angular momenta (vectors) are available.

`logical angularMomentaAvailable()` Return true if angular momenta (magnitudes) are available.

`logical angularMomentaIncludeSubhalos()` Returns a Boolean specifying whether halo angular momenta (or spins) include the contribution from their subhalos.

`void autoHook()` Insert any event hooks required by this object.

`logical canReadSubsets()` Returns true if arbitrary subsets of halos from a forest can be read.

`void close()` Closes the file.

`double precision cubeLength(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcol integer\textgreater} status\argout)` Returns the length of the simulation cube.

`void deepCopy(\textcolor{red}{\textless class((mergerTreeImporterClass))\textgreater} destination\argin` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`void import(\textcolor{red}{\textless integer\textgreater} i\argin,\textcolor{red}{\textless class((nodeDataMinimal))\textgreater} nodes\argout,\textcolor{red}{\textless integer((c_-size_t))\textgreater} nodeSubset\argin,\textcolor{red}{\textless logical\textgreater} requireScaleRadii\argin,\textcolor{red}{\textless logical\textgreater} requireAngularMomenta\argin,logical\textgreater} requireAngularMomenta3D\argin,\textcolor{red}{\textless logical\textgreater} requireSpin\argin,\textcolor{red}{\textless logical\textgreater} requireSpin3D\argin,\textcolor{redlogical\textgreater} requirePositions\argin,\textcolor{red}{\textless logical\textgreater} structureOnly\argin,\textcolor{red}{\textless type((varying_string))\textgreater} requireNamedReals\argin,\textcolor{red}{\textless type((varying_string))\textgreater} requireNamedIntegers\argin)` Imports the i^{th} tree.

`<logical> isDefault()` Return true if this is the default object of this class.

`logical massesIncludeSubhalos()` Returns a Boolean specifying whether halo masses include the contribution from their subhalos.

`integer(kind=c_size_t) nodeCount(\textcolor{red}{\textless integer\textgreater} i\argin)` Returns the number of nodes in the i^{th} tree.

`type(varying_string) objectType()` Return the type of the object.

`void open(\textcolor{red}{\textless type((varying_string))\textgreater} fileName\argin)` Opens the file.

`logical particleCountAvailable()` Return true if particle counts are available.

`integer positionsArePeriodic()` Returns a Boolean integer specifying whether positions are periodic.

`logical positionsAvailable(\textcolor{red}{\textless logical\textgreater} positions\argin,\textcolor{relogical\textgreater} velocities\argin)` Return true if positions and/or velocities are available.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`logical scaleRadiiAvailable()` Return true if scale radii are available.

`logical spin3DAvailable()` Return true if spin (vectors) are available.

`logical spinAvailable()` Return true if spin (magnitudes) are available.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textltype((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textlestype((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`void subhaloTrace(\textcolor{red}{\textless class((nodeData))\textgreater} node\argin,\textcolor{red}{\double precision\textgreater} time\argout,\textcolor{red}{\textless double precision\textgreater} position\argout,\textcolor{red}{\textless double precision\textgreater} velocity\argout)` Supplies epochs, positions, and velocities for traced subhalos.

integer(kind=c_size_t) subhaloTraceCount(\textcolor{red}{\textless class((nodeData))\textgreater} node\argin) Returns the length of a node's subhalo trace.

integer(kind=c_size_t) treeCount() Returns a count of the number of trees available.

integer(kind=kind_int8) treeIndex(\textcolor{red}{\textless integer\textgreater} i\argin) Returns the index of the $i^{\rm th}$ tree.

integer treesAreSelfContained() Returns a Boolean integer specifying whether trees are self-contained.

integer treesHaveSubhalos() Returns a Boolean integer specifying whether or not the trees have subhalos.

double precision treeWeight(\textcolor{red}{\textless integer\textgreater} i\argin) Returns the weight to assign to the $i^{\rm th}$ tree.

integer velocitiesIncludeHubbleFlow() Returns a Boolean integer specifying whether velocities include the Hubble flow.

logical velocityDispersionAvailable() Return true if halo velocity dispersions are available.

logical velocityMaximumAvailable() Return true if rotation curve velocity maxima are available.

mergerTreeImporterSussing

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

logical angularMomenta3DAvailable() Return true if angular momenta (vectors) are available.

logical angularMomentaAvailable() Return true if angular momenta (magnitudes) are available.

logical angularMomentaIncludeSubhalos() Returns a Boolean specifying whether halo angular momenta (or spins) include the contribution from their subhalos.

void autoHook() Insert any event hooks required by this object.

logical canReadSubsets() Returns true if arbitrary subsets of halos from a forest can be read.

void close() Closes the file.

double precision cubeLength(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcol integer\textgreater} status\argout) Returns the length of the simulation cube.

void deepCopy(\textcolor{red}{\textless class((mergerTreeImporterClass))\textgreater} destination\argin Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

void import(\textcolor{red}{\textless integer\textgreater} i\argin,\textcolor{red}{\textless class((nodeDataMinimal))\textgreater} nodes\argout,\textcolor{red}{\textless integer((c_-size_t))\textgreater} nodeSubset\argin,\textcolor{red}{\textless logical\textgreater} requireScaleRadii\argin,\textcolor{red}{\textless logical\textgreater} requireAngularMomenta\argin, logical\textgreater} requireAngularMomenta3D\argin,\textcolor{red}{\textless logical\textgreater} requireSpin\argin,\textcolor{red}{\textless logical\textgreater} requireSpin3D\argin,\textcolor{red logical\textgreater} requirePositions\argin,\textcolor{red}{\textless logical\textgreater} structureOnly\argin,\textcolor{red}{\textless type((varying_string))\textgreater} requireNamedReals\argin,\textcolor{red}{\textless type((varying_string))\textgreater} requireNamedIntegers\argin) Imports the i^{th} tree.

<logical> inSubvolume(<double> x→, <double> y→, <double> z→, <logical> [buffered]→) Return true if the given x,y,z position lies within the current subvolume (plus the buffer region if buffered is true.

<logical> inSubvolume1D(<double> x→, <integer> iSubvolume→, <logical> [buffered]→) Return true if the given x position lies within the iSubvolume$^{\text{th}}$ subvolume (plus the buffer region if buffered is true.

<logical> isDefault() Return true if this is the default object of this class.

<logical> load(<integer(kind_int8,:)> nodeSelfIndices←, <integer(c_size_t,:)> nodeIndexRanks←, <integer(c_size_t,:)> nodeDescendentLocations←, <logical> nodeIncomplete←, <integer(c_-size_t)> nodeCountTrees←, <integer(kind_int8,:)> nodeTreeIndices←, <logical> treeIndicesAssigned <logical> branchJumpCheckRequired←, <type(importerUnits)> massUnits←, <type(importerUnits)> lengthUnits←, <type(importerUnits)> velocityUnits←) Load the halo data.

logical massesIncludeSubhalos() Returns a Boolean specifying whether halo masses include the contribution from their subhalos.

integer(kind=c_size_t) nodeCount(\textcolor{red}{\textless integer\textgreater} i\argin) Returns the number of nodes in the i^{th} tree.

type(varying_string) objectType() Return the type of the object.

void open(\textcolor{red}{\textless type((varying_string))\textgreater} fileName\argin) Opens the file.

logical particleCountAvailable() Return true if particle counts are available.

integer positionsArePeriodic() Returns a Boolean integer specifying whether positions are periodic.

logical positionsAvailable(\textcolor{red}{\textless logical\textgreater} positions\argin,\textcolor{re logical\textgreater} velocities\argin) Return true if positions and/or velocities are available.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

logical scaleRadiiAvailable() Return true if scale radii are available.

`logical spin3DAvailable()` Return true if spin (vectors) are available.

`logical spinAvailable()` Return true if spin (magnitudes) are available.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`void subhaloTrace(\textcolor{red}{\textless class((nodeData))\textgreater} node\argin,\textcolor{red}{\ double precision\textgreater} time\argout,\textcolor{red}{\textless double precision\textgreater} position\argout,\textcolor{red}{\textless double precision\textgreater} velocity\argout)` Supplies epochs, positions, and velocities for traced subhalos.

`integer(kind=c_size_t) subhaloTraceCount(\textcolor{red}{\textless class((nodeData))\textgreater} node\argin)` Returns the length of a node's subhalo trace.

`integer(kind=c_size_t) treeCount()` Returns a count of the number of trees available.

`integer(kind=kind_int8) treeIndex(\textcolor{red}{\textless integer\textgreater} i\argin)` Returns the index of the $i^{\rm th}$ tree.

`integer treesAreSelfContained()` Returns a Boolean integer specifying whether trees are self-contained.

`integer treesHaveSubhalos()` Returns a Boolean integer specifying whether or not the trees have subhalos.

`double precision treeWeight(\textcolor{red}{\textless integer\textgreater} i\argin)` Returns the weight to assign to the $i^{\rm th}$ tree.

`<logical> valueIsBad(<double> x→)` Return true if the given `x` value is bad.

`integer velocitiesIncludeHubbleFlow()` Returns a Boolean integer specifying whether velocities include the Hubble flow.

`logical velocityDispersionAvailable()` Return true if halo velocity dispersions are available.

`logical velocityMaximumAvailable()` Return true if rotation curve velocity maxima are available.

mergerTreeImporterSussingASCII

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`logical angularMomenta3DAvailable()` Return true if angular momenta (vectors) are available.

`logical angularMomentaAvailable()` Return true if angular momenta (magnitudes) are available.

`logical angularMomentaIncludeSubhalos()` Returns a Boolean specifying whether halo angular momenta (or spins) include the contribution from their subhalos.

`void autoHook()` Insert any event hooks required by this object.

`logical canReadSubsets()` Returns true if arbitrary subsets of halos from a forest can be read.

`void close()` Closes the file.

`double precision cubeLength(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcol integer\textgreater} status\argout)` Returns the length of the simulation cube.

`void deepCopy(\textcolor{red}{\textless class((mergerTreeImporterClass))\textgreater} destination\argin` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`void import(\textcolor{red}{\textless integer\textgreater} i\argin,\textcolor{red}{\textless class((nodeDataMinimal))\textgreater} nodes\argout,\textcolor{red}{\textless integer((c_size_t))\textgreater} nodeSubset\argin,\textcolor{red}{\textless logical\textgreater} requireScaleRadii\argin,\textcolor{red}{\textless logical\textgreater} requireAngularMomenta\argin, logical\textgreater} requireAngularMomenta3D\argin,\textcolor{red}{\textless logical\textgreater} requireSpin\argin,\textcolor{red}{\textless logical\textgreater} requireSpin3D\argin,\textcolor{red logical\textgreater} requirePositions\argin,\textcolor{red}{\textless logical\textgreater} structureOnly\argin,\textcolor{red}{\textless type((varying_string))\textgreater} requireNamedReals\argin,\textcolor{red}{\textless type((varying_string))\textgreater} requireNamedIntegers\argin)` Imports the i^{th} tree.

`<logical> inSubvolume(<double> x→, <double> y→, <double> z→, <logical> [buffered]→)` Return true if the given `x,y,z` position lies within the current subvolume (plus the buffer region if `buffered` is true.

`<logical> inSubvolume1D(<double> x→, <integer> iSubvolume→, <logical> [buffered]→)` Return true if the given `x` position lies within the `iSubvolume`$^{\text{th}}$ subvolume (plus the buffer region if `buffered` is true.

`<logical> isDefault()` Return true if this is the default object of this class.

`<logical> load(<integer(kind_int8,:)> nodeSelfIndices←, <integer(c_size_t,:)> nodeIndexRanks←, <integer(c_size_t,:)> nodeDescendentLocations←, <logical> nodeIncomplete←, <integer(c_size_t)> nodeCountTrees←, <integer(kind_int8,:)> nodeTreeIndices←, <logical> treeIndicesAssigned <logical> branchJumpCheckRequired←, <type(importerUnits)> massUnits←, <type(importerUnits)> lengthUnits←, <type(importerUnits)> velocityUnits←)` Load the halo data.

`logical massesIncludeSubhalos()` Returns a Boolean specifying whether halo masses include the contribution from their subhalos.

`integer(kind=c_size_t) nodeCount(\textcolor{red}{\textless integer\textgreater} i\argin)` Returns the number of nodes in the i^{th} tree.

`type(varying_string) objectType()` Return the type of the object.

`void open(\textcolor{red}{\textless type((varying_string))\textgreater} fileName\argin)` Opens the file.

`logical particleCountAvailable()` Return true if particle counts are available.

`integer positionsArePeriodic()` Returns a Boolean integer specifying whether positions are periodic.

`logical positionsAvailable(\textcolor{red}{\textless logical\textgreater} positions\argin,\textcolor{re logical\textgreater} velocities\argin)` Return true if positions and/or velocities are available.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`logical scaleRadiiAvailable()` Return true if scale radii are available.

`logical spin3DAvailable()` Return true if spin (vectors) are available.

`logical spinAvailable()` Return true if spin (magnitudes) are available.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`void subhaloTrace(\textcolor{red}{\textless class((nodeData))\textgreater} node\argin,\textcolor{red}{\ double precision\textgreater} time\argout,\textcolor{red}{\textless double precision\textgreater} position\argout,\textcolor{red}{\textless double precision\textgreater} velocity\argout)` Supplies epochs, positions, and velocities for traced subhalos.

`integer(kind=c_size_t) subhaloTraceCount(\textcolor{red}{\textless class((nodeData))\textgreater} node\argin)` Returns the length of a node's subhalo trace.

`integer(kind=c_size_t) treeCount()` Returns a count of the number of trees available.

`integer(kind=kind_int8) treeIndex(\textcolor{red}{\textless integer\textgreater} i\argin)` Returns the index of the $i^{\rm th}$ tree.

`integer treesAreSelfContained()` Returns a Boolean integer specifying whether trees are self-contained.

`integer treesHaveSubhalos()` Returns a Boolean integer specifying whether or not the trees have subhalos.

`double precision treeWeight(\textcolor{red}{\textless integer\textgreater} i\argin)` Returns the weight to assign to the $i^{\rm th}$ tree.

`<logical> valueIsBad(<double> x→)` Return true if the given `x` value is bad.

`integer velocitiesIncludeHubbleFlow()` Returns a Boolean integer specifying whether velocities include the Hubble flow.

`logical velocityDispersionAvailable()` Return true if halo velocity dispersions are available.

`logical velocityMaximumAvailable()` Return true if rotation curve velocity maxima are available.

mergerTreeImporterSussingHDF5

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`logical angularMomenta3DAvailable()` Return true if angular momenta (vectors) are available.

`logical angularMomentaAvailable()` Return true if angular momenta (magnitudes) are available.

`logical angularMomentaIncludeSubhalos()` Returns a Boolean specifying whether halo angular momenta (or spins) include the contribution from their subhalos.

`void autoHook()` Insert any event hooks required by this object.

`logical canReadSubsets()` Returns true if arbitrary subsets of halos from a forest can be read.

`void close()` Closes the file.

`double precision cubeLength(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcol integer\textgreater} status\argout)` Returns the length of the simulation cube.

`void deepCopy(\textcolor{red}{\textless class((mergerTreeImporterClass))\textgreater} destination\argin` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`void import(\textcolor{red}{\textless integer\textgreater} i\argin,\textcolor{red}{\textless class((nodeDataMinimal))\textgreater} nodes\argout,\textcolor{red}{\textless integer((c_-size_t))\textgreater} nodeSubset\argin,\textcolor{red}{\textless logical\textgreater} requireScaleRadii\argin,\textcolor{red}{\textless logical\textgreater} requireAngularMomenta\argin, logical\textgreater} requireAngularMomenta3D\argin,\textcolor{red}{\textless logical\textgreater} requireSpin\argin,\textcolor{red}{\textless logical\textgreater} requireSpin3D\argin,\textcolor{red logical\textgreater} requirePositions\argin,\textcolor{red}{\textless logical\textgreater} structureOnly\argin,\textcolor{red}{\textless type((varying_string))\textgreater} requireNamedReals\argin,\textcolor{red}{\textless type((varying_string))\textgreater} requireNamedIntegers\argin)` Imports the i^{th} tree.

`<logical> inSubvolume(<double> x→, <double> y→, <double> z→, <logical> [buffered]→)` Return true if the given `x,y,z` position lies within the current subvolume (plus the buffer region if `buffered` is true.

`<logical> inSubvolume1D(<double> x→, <integer> iSubvolume→, <logical> [buffered]→)` Return true if the given `x` position lies within the `iSubvolume`$^{\text{th}}$ subvolume (plus the buffer region if `buffered` is true.

`<logical> isDefault()` Return true if this is the default object of this class.

<logical> load(<integer(kind_int8,:)> nodeSelfIndices←, <integer(c_size_t,:)> nodeIndexRanks←, <integer(c_size_t,:)> nodeDescendentLocations←, <logical> nodeIncomplete←, <integer(c_size_t)> nodeCountTrees←, <integer(kind_int8,:)> nodeTreeIndices←, <logical> treeIndicesAssigned←, <logical> branchJumpCheckRequired←, <type(importerUnits)> massUnits←, <type(importerUnits)> lengthUnits←, <type(importerUnits)> velocityUnits←) Load the halo data.

logical massesIncludeSubhalos() Returns a Boolean specifying whether halo masses include the contribution from their subhalos.

integer(kind=c_size_t) nodeCount(\textcolor{red}{\textless integer\textgreater} i\argin) Returns the number of nodes in the $i^{\rm th}$ tree.

type(varying_string) objectType() Return the type of the object.

void open(\textcolor{red}{\textless type((varying_string))\textgreater} fileName\argin) Opens the file.

logical particleCountAvailable() Return true if particle counts are available.

integer positionsArePeriodic() Returns a Boolean integer specifying whether positions are periodic.

logical positionsAvailable(\textcolor{red}{\textless logical\textgreater} positions\argin,\textcolor{red}{\textless logical\textgreater} velocities\argin) Return true if positions and/or velocities are available.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

logical scaleRadiiAvailable() Return true if scale radii are available.

logical spin3DAvailable() Return true if spin (vectors) are available.

logical spinAvailable() Return true if spin (magnitudes) are available.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

void subhaloTrace(\textcolor{red}{\textless class((nodeData))\textgreater} node\argin,\textcolor{red}{\textless double precision\textgreater} time\argout,\textcolor{red}{\textless double precision\textgreater} position\argout,\textcolor{red}{\textless double precision\textgreater} velocity\argout) Supplies epochs, positions, and velocities for traced subhalos.

integer(kind=c_size_t) subhaloTraceCount(\textcolor{red}{\textless class((nodeData))\textgreater} node\argin) Returns the length of a node's subhalo trace.

integer(kind=c_size_t) treeCount() Returns a count of the number of trees available.

integer(kind=kind_int8) treeIndex(\textcolor{red}{\textless integer\textgreater} i\argin) Returns the index of the i^{th} tree.

integer treesAreSelfContained() Returns a Boolean integer specifying whether trees are self-contained.

integer treesHaveSubhalos() Returns a Boolean integer specifying whether or not the trees have subhalos.

double precision treeWeight(\textcolor{red}{\textless integer\textgreater} i\argin) Returns the weight to assign to the i^{th} tree.

<logical> valueIsBad(<double> x→) Return true if the given x value is bad.

integer velocitiesIncludeHubbleFlow() Returns a Boolean integer specifying whether velocities include the Hubble flow.

logical velocityDispersionAvailable() Return true if halo velocity dispersions are available.

logical velocityMaximumAvailable() Return true if rotation curve velocity maxima are available.

mergerTreeMassResolutionClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((mergerTreeMassResolutionClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

double precision resolution(\textcolor{red}{\textless type((mergerTree))\textgreater} tree\argin) Gives the mass resolution to use for the given tree.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerTreeMassResolutionFixed

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((mergerTreeMassResolutionClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

double precision resolution(\textcolor{red}{\textless type((mergerTree))\textgreater} tree\argin)
Gives the mass resolution to use for the given tree.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerTreeMassResolutionScaled

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((mergerTreeMassResolutionClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

double precision resolution(\textcolor{red}{\textless type((mergerTree))\textgreater} tree\argin)
Gives the mass resolution to use for the given tree.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerTreeNodeEvolverClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((mergerTreeNodeEvolverClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

void evolve(\textcolor{red}{\textless type((mergerTree))\textgreater} tree \arginout,\textcolor{red}{\
type((treeNode))\textgreater} node \arginout,\textcolor{red}{\textless double precision\textgreate
timeEnd \argin,\textcolor{red}{\textless logical\textgreater} interrupted \argout,\textcolor{red}{\
procedure((interruptTask))\textgreater} functionInterrupt \argout,\textcolor{red}{\textless
class((galacticStructureSolverClass))\textgreater} galacticStructureSolver__\argin,\textcolor{red}{
integer((kind_int8))\textgreater} systemClockMaximum \argin,\textcolor{red}{\textless
integer\textgreater} status \argout) Evolve a node merger tree.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

logical isAccurate(\textcolor{red}{\textless double precision\textgreater} valueNode\argin,\textcolor{r
double precision\textgreater} valueExpected\argin) Return true if a tree node property is within expected accuracy of a given value.

<logical> isDefault() Return true if this is the default object of this class.

void merge(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Handles instances where node is about to merge with its parent node.

type(varying_string) objectType() Return the type of the object.

void promote(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Promote node to its parent node, then destroy it.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerTreeNodeEvolverStandard

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((mergerTreeNodeEvolverClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

void evolve(\textcolor{red}{\textless type((mergerTree))\textgreater} tree \arginout,\textcolor{red}{\ type((treeNode))\textgreater} node \arginout,\textcolor{red}{\textless double precision\textgreate timeEnd \argin,\textcolor{red}{\textless logical\textgreater} interrupted \argout,\textcolor{red}{\ procedure((interruptTask))\textgreater} functionInterrupt \argout,\textcolor{red}{\textless class((galacticStructureSolverClass))\textgreater} galacticStructureSolver__\argin,\textcolor{red}{ integer((kind_int8))\textgreater} systemClockMaximum \argin,\textcolor{red}{\textless integer\textgreater} status \argout) Evolve a node merger tree.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

logical isAccurate(\textcolor{red}{\textless double precision\textgreater} valueNode\argin,\textcolor{r double precision\textgreater} valueExpected\argin) Return true if a tree node property is within expected accuracy of a given value.

<logical> isDefault() Return true if this is the default object of this class.

void merge(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Handles instances where node is about to merge with its parent node.

type(varying_string) objectType() Return the type of the object.

void promote(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Promote node to its parent node, then destroy it.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerTreeNodeMergerClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((mergerTreeNodeMergerClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

void process(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Process the merger between node and its parent node, then destroy it.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerTreeNodeMergerSingleLevelHierarchy

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((mergerTreeNodeMergerClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

void process(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Process the merger between node and its parent node, then destroy it.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerTreeOperatorAssignOrbits

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((mergerTreeOperatorClass))\textgreater} destination\argin` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`void finalize()` Finalize a merger tree operator.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`void operate(\textcolor{red}{\textless type((mergerTree))\textgreater} tree\arginout)` Perform an operation on the merger tree.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

mergerTreeOperatorAugment

`<integer> acceptTree(<type(treeNode)> *node↔,<type(mergerTree)> tree↔,<integer>nodeChildCount↔,<logi treeBest↔,<double>treeBestWorstFit↔,<logical>treeBestOverride→,<double>massCutoffScale↔,<logica newTreeBest↔)` Determine if a newly built tree is an acceptable match.

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`<integer> buildTreeFromNode(<type(treeNode)> *node↔,<logical>extendingEndNode→,<double>tolerance→,<d treeBest↔,<double>treeBestWorstFit↔,<logical>treeBestOverride→,<double>massCutoffScale↔,<double` Build a merger tree starting from the given node.

`void deepCopy(\textcolor{red}{\textless class((mergerTreeOperatorClass))\textgreater} destination\argin` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

<void> extendNonOverlapNodes(<type(mergerTree)> tree↔,<type(treeNode)> *nodeNonOverlapFirst↔,<double> treeBest↔,<double>massCutoffScale↔) Graft new branches onto all end-nodes of a newly built tree.

void finalize() Finalize a merger tree operator.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

<void> nonOverlapReinsert(<type(treeNode)> *listFirstElement↔) Reinsert a linked list of non-overlap nodes into their parent tree.

type(varying_string) objectType() Return the type of the object.

void operate(\textcolor{red}{\textless type((mergerTree))\textgreater} tree\arginout) Perform an operation on the merger tree.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

<void> sortChildren(<type(treeNode)> *node↔) Sort child nodes into descending mass order.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerTreeOperatorClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((mergerTreeOperatorClass))\textgreater} destination\argin Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

void finalize() Finalize a merger tree operator.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

void operate(\textcolor{red}{\textless type((mergerTree))\textgreater} tree\arginout) Perform an operation on the merger tree.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerTreeOperatorConditionalMF

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

<double(:)> binWeights(<double> mass→,<double> time→,<double> massLogarithmicMinimumBins→,<double> massLogarithmicWidthInverseBins→,<integer> countBins→) Compute weights for a halo in each bin of the mass function.

<double(:,:)> binWeights2D(<double> mass1→,<double> time1→,<double> mass2→,<double> time2→,<double> massLogarithmicMinimumBins1→,<double> massLogarithmicWidthInverseBins1→,<integer> countBins2→,<double> massLogarithmicMinimumBins2→,<double> massLogarithmicWidthInverseBins2→,<in countBins2→) Compute weights for a halo in each bin of a 2D mass function.

void deepCopy(\textcolor{red}{\textless class((mergerTreeOperatorClass))\textgreater} destination\argin Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

void finalize() Finalize a merger tree operator.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

`void operate(\textcolor{red}{\textless type((mergerTree))\textgreater} tree\arginout)` Perform an operation on the merger tree.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

mergerTreeOperatorDeforest

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((mergerTreeOperatorClass))\textgreater} destination\argin` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`void finalize()` Finalize a merger tree operator.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`void operate(\textcolor{red}{\textless type((mergerTree))\textgreater} tree\arginout)` Perform an operation on the merger tree.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

mergerTreeOperatorDumpToGraphViz

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((mergerTreeOperatorClass))\textgreater} destination\argin` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`void finalize()` Finalize a merger tree operator.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`void operate(\textcolor{red}{\textless type((mergerTree))\textgreater} tree\arginout)` Perform an operation on the merger tree.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

mergerTreeOperatorExport

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((mergerTreeOperatorClass))\textgreater} destination\argin` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`void finalize()` Finalize a merger tree operator.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`void operate(\textcolor{red}{\textless type((mergerTree))\textgreater} tree\arginout)` Perform an operation on the merger tree.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

mergerTreeOperatorInformationContent

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((mergerTreeOperatorClass))\textgreater} destination\argir` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`void finalize()` Finalize a merger tree operator.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`void operate(\textcolor{red}{\textless type((mergerTree))\textgreater} tree\arginout)` Perform an operation on the merger tree.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerTreeOperatorMassAccretionHistory

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((mergerTreeOperatorClass))\textgreater} destination\argin Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

void finalize() Finalize a merger tree operator.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

void operate(\textcolor{red}{\textless type((mergerTree))\textgreater} tree\arginout) Perform an operation on the merger tree.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerTreeOperatorMonotonizeMassGrowth

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((mergerTreeOperatorClass))\textgreater} destination\argin Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

void finalize() Finalize a merger tree operator.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

void operate(\textcolor{red}{\textless type((mergerTree))\textgreater} tree\arginout) Perform an operation on the merger tree.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerTreeOperatorNull

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((mergerTreeOperatorClass))\textgreater} destination\argin Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

`void finalize()` Finalize a merger tree operator.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`void operate(\textcolor{red}{\textless type((mergerTree))\textgreater} tree\arginout)` Perform an operation on the merger tree.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

mergerTreeOperatorOutputRootMasses

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((mergerTreeOperatorClass))\textgreater} destination\argir` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`void finalize()` Finalize a merger tree operator.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`void operate(\textcolor{red}{\textless type((mergerTree))\textgreater} tree\arginout)` Perform an operation on the merger tree.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerTreeOperatorParticulate

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((mergerTreeOperatorClass))\textgreater} destination\argin Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

void finalize() Finalize a merger tree operator.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

void operate(\textcolor{red}{\textless type((mergerTree))\textgreater} tree\arginout) Perform an operation on the merger tree.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerTreeOperatorPerturbMasses

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((mergerTreeOperatorClass))\textgreater} destination\argin
Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

void finalize() Finalize a merger tree operator.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

void operate(\textcolor{red}{\textless type((mergerTree))\textgreater} tree\arginout) Perform an operation on the merger tree.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerTreeOperatorProfiler

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((mergerTreeOperatorClass))\textgreater} destination\argin
Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

`void finalize()` Finalize a merger tree operator.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`void operate(\textcolor{red}{\textless type((mergerTree))\textgreater} tree\arginout)` Perform an operation on the merger tree.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl` `type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-` `size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles` `type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-` `size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

mergerTreeOperatorPruneBaryons

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\` `character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((mergerTreeOperatorClass))\textgreater} destination\argir` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex` `logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`void finalize()` Finalize a merger tree operator.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`void operate(\textcolor{red}{\textless type((mergerTree))\textgreater} tree\arginout)` Perform an operation on the merger tree.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerTreeOperatorPruneBranchTips

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((mergerTreeOperatorClass))\textgreater} destination\argin Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

void finalize() Finalize a merger tree operator.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

void operate(\textcolor{red}{\textless type((mergerTree))\textgreater} tree\arginout) Perform an operation on the merger tree.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerTreeOperatorPruneByMass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((mergerTreeOperatorClass))\textgreater} destination\argin Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

void finalize() Finalize a merger tree operator.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

void operate(\textcolor{red}{\textless type((mergerTree))\textgreater} tree\arginout) Perform an operation on the merger tree.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerTreeOperatorPruneByTime

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((mergerTreeOperatorClass))\textgreater} destination\argin Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

`void finalize()` Finalize a merger tree operator.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`void operate(\textcolor{red}{\textless type((mergerTree))\textgreater} tree\arginout)` Perform an operation on the merger tree.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

mergerTreeOperatorPruneClones

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((mergerTreeOperatorClass))\textgreater} destination\argi` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`void finalize()` Finalize a merger tree operator.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`void operate(\textcolor{red}{\textless type((mergerTree))\textgreater} tree\arginout)` Perform an operation on the merger tree.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerTreeOperatorPruneHierarchy

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((mergerTreeOperatorClass))\textgreater} destination\argin Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

void finalize() Finalize a merger tree operator.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

void operate(\textcolor{red}{\textless type((mergerTree))\textgreater} tree\arginout) Perform an operation on the merger tree.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerTreeOperatorPruneLightcone

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((mergerTreeOperatorClass))\textgreater} destination\argin
Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

void finalize() Finalize a merger tree operator.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

void operate(\textcolor{red}{\textless type((mergerTree))\textgreater} tree\arginout) Perform an operation on the merger tree.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerTreeOperatorPruneNonEssential

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((mergerTreeOperatorClass))\textgreater} destination\argin
Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

`void finalize()` Finalize a merger tree operator.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`void operate(\textcolor{red}{\textless type((mergerTree))\textgreater} tree\arginout)` Perform an operation on the merger tree.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

mergerTreeOperatorRegridTimes

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((mergerTreeOperatorClass))\textgreater} destination\argi` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`void finalize()` Finalize a merger tree operator.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`void operate(\textcolor{red}{\textless type((mergerTree))\textgreater} tree\arginout)` Perform an operation on the merger tree.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerTreeOperatorSelectWithinRange

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((mergerTreeOperatorClass))\textgreater} destination\argin Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

void finalize() Finalize a merger tree operator.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

void operate(\textcolor{red}{\textless type((mergerTree))\textgreater} tree\arginout) Perform an operation on the merger tree.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerTreeOperatorSequence

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((mergerTreeOperatorClass))\textgreater} destination\argin Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

void finalize() Finalize a merger tree operator.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

void operate(\textcolor{red}{\textless type((mergerTree))\textgreater} tree\arginout) Perform an operation on the merger tree.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerTreeOutputterAnalyzer

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((mergerTreeOutputterClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

`void finalize()` Finalize output of merger trees.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges`
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`void output(\textcolor{red}{\textless type((mergerTree))\textgreater} tree \arginout,\textcolor{red}{\t`
`integer((c_size_t ))\textgreater} indexOutput \argin,\textcolor{red}{\textless double`
`precision\textgreater} time \argin,\textcolor{red}{\textless logical\textgreater} isLastOutput\argi`
Output a merger tree.

`void reduce(\textcolor{red}{\textless class((mergerTreeOutputterClass))\textgreater} reduced\arginout)`
Reduce the object onto another object of the class.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl`
`type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-`
`size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles`
`type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-`
`size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

mergerTreeOutputterClass

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\`
`character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((mergerTreeOutputterClass))\textgreater}`
`destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex`
`logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`void finalize()` Finalize output of merger trees.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges`
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`void output(\textcolor{red}{\textless type((mergerTree))\textgreater} tree \arginout,\textcolor{red}{\t integer((c_size_t ))\textgreater} indexOutput \argin,\textcolor{red}{\textless double precision\textgreater} time \argin,\textcolor{red}{\textless logical\textgreater} isLastOutput\argi` Output a merger tree.

`void reduce(\textcolor{red}{\textless class((mergerTreeOutputterClass))\textgreater} reduced\arginout)` Reduce the object onto another object of the class.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

mergerTreeOutputterMulti

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((mergerTreeOutputterClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`void finalize()` Finalize output of merger trees.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`void output(\textcolor{red}{\textless type((mergerTree))\textgreater} tree \arginout,\textcolor{red}{\t integer((c_size_t ))\textgreater} indexOutput \argin,\textcolor{red}{\textless double precision\textgreater} time \argin,\textcolor{red}{\textless logical\textgreater} isLastOutput\argi` Output a merger tree.

`void reduce(\textcolor{red}{\textless class((mergerTreeOutputterClass))\textgreater} reduced\arginout)` Reduce the object onto another object of the class.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerTreeOutputterNull

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((mergerTreeOutputterClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

void finalize() Finalize output of merger trees.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

void output(\textcolor{red}{\textless type((mergerTree))\textgreater} tree \arginout,\textcolor{red}{\t integer((c_size_t))\textgreater} indexOutput \argin,\textcolor{red}{\textless double precision\textgreater} time \argin,\textcolor{red}{\textless logical\textgreater} isLastOutput\argi Output a merger tree.

void reduce(\textcolor{red}{\textless class((mergerTreeOutputterClass))\textgreater} reduced\arginout) Reduce the object onto another object of the class.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerTreeOutputterStandard

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void buffersAllocate(<integer(c_size_t)> indexOutput→) Allocate buffers for storage of properties.

void deepCopy(\textcolor{red}{\textless class((mergerTreeOutputterClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

void dumpDoubleBuffer(<integer(c_size_t)> indexOutput→) Dump the contents of the double properties buffer to the Galacticus output file.

void dumpIntegerBuffer(<integer(c_size_t)> indexOutput→) Dump the contents of the integer properties buffer to the Galacticus output file.

void extendDoubleBuffer() Extend the size of the double buffer.

void extendIntegerBuffer() Extend the size of the integer buffer.

void finalize() Finalize output of merger trees.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

void makeGroup(<type(mergerTree)> tree↔, <integer(c_size_t)> indexOutput→) Make an group in the Galacticus file in which to store tree.

type(varying_string) objectType() Return the type of the object.

void output(\textcolor{red}{\textless type((mergerTree))\textgreater} tree \arginout,\textcolor{red}{\t integer((c_size_t))\textgreater} indexOutput \argin,\textcolor{red}{\textless double precision\textgreater} time \argin,\textcolor{red}{\textless logical\textgreater} isLastOutput\argi Output a merger tree.

void outputGroupCreate(<integer(c_size_t)> indexOutput→, <double> time→) Create a group in which to store this output.

void propertiesCount(<double> time→, <type(treeNode)> *node↔) Count up the number of properties that will be output.

`void propertyNamesEstablish(<double> time→, <type(treeNode)> *node↔)` Set names for the properties.

`void reduce(\textcolor{red}{\textless class((mergerTreeOutputterClass))\textgreater} reduced\arginout)` Reduce the object onto another object of the class.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

mergerTreeWalkerAllAndFormationNodes

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((mergerTreeWalkerClass))\textgreater} destination\arginou` Perform a deep copy of the object.

`<void> descend()` Descend through the hierarchy to the deepest node along the current branch.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`logical next(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Update the pointer to the next node to visit. Returns true if such a node exists, returns false if no such node exists (i.e. if all nodes have been visited already).

`logical nodesRemain()` Returns true if more nodes remain to be walked to in the tree.

`type(varying_string) objectType()` Return the type of the object.

`<void> previous(<type(treeNode)> *node↔)` Step back to the previously visited node (if possible).

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

<void> setNode(<type(treeNode)> node→) Set the walker to the given node.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerTreeWalkerAllNodes

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((mergerTreeWalkerClass))\textgreater} destination\arginou Perform a deep copy of the object.

<void> descend() Descend through the hierarchy to the deepest node along the current branch.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

logical next(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Update the pointer to the next node to visit. Returns true if such a node exists, returns false if no such node exists (i.e. if all nodes have been visited already).

logical nodesRemain() Returns true if more nodes remain to be walked to in the tree.

type(varying_string) objectType() Return the type of the object.

<void> previous(<type(treeNode)> *node↔) Step back to the previously visited node (if possible).

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

<void> setNode(<type(treeNode)> node→) Set the walker to the given node.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerTreeWalkerAllNodesBranch

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((mergerTreeWalkerClass))\textgreater} destination\arginou Perform a deep copy of the object.

<void> descend() Descend through the hierarchy to the deepest node along the current branch.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

logical next(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Update the pointer to the next node to visit. Returns true if such a node exists, returns false if no such node exists (i.e. if all nodes have been visited already).

logical nodesRemain() Returns true if more nodes remain to be walked to in the tree.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerTreeWalkerClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((mergerTreeWalkerClass))\textgreater} destination\arginou Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

logical next(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Update the pointer to the next node to visit. Returns true if such a node exists, returns false if no such node exists (i.e. if all nodes have been visited already).

logical nodesRemain() Returns true if more nodes remain to be walked to in the tree.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerTreeWalkerIsolatedNodes

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((mergerTreeWalkerClass))\textgreater} destination\arginou Perform a deep copy of the object.

<void> descend() Descend through the hierarchy to the deepest node along the current branch.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`logical next(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Update the pointer to the next node to visit. Returns true if such a node exists, returns false if no such node exists (i.e. if all nodes have been visited already).

`logical nodesRemain()` Returns true if more nodes remain to be walked to in the tree.

`type(varying_string) objectType()` Return the type of the object.

`<void> previous(<type(treeNode)> *node↔)` Step back to the previously visited node (if possible).

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`<void> setNode(<type(treeNode)> node→)` Set the walker to the given node.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

mergerTreeWalkerIsolatedNodesBranch

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((mergerTreeWalkerClass))\textgreater} destination\arginou` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

logical next(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Update the pointer to the next node to visit. Returns true if such a node exists, returns false if no such node exists (i.e. if all nodes have been visited already).

logical nodesRemain() Returns true if more nodes remain to be walked to in the tree.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

mergerTreeWalkerTreeConstruction

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((mergerTreeWalkerClass))\textgreater} destination\arginou Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

logical next(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Update the pointer to the next node to visit. Returns true if such a node exists, returns false if no such node exists (i.e. if all nodes have been visited already).

logical nodesRemain() Returns true if more nodes remain to be walked to in the tree.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

metaTreeProcessingTimeClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((metaTreeProcessingTimeClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision time(\textcolor{red}{\textless double precision\textgreater} massTree\argin) Return an estimate of the time needed to process a tree of the given mass.

metaTreeProcessingTimeFile

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((metaTreeProcessingTimeClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision time(\textcolor{red}{\textless double precision\textgreater} massTree\argin)
Return an estimate of the time needed to process a tree of the given mass.

modelParameterActive

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((modelParameterClass))\textgreater} destination\arginout)
Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision logPrior(\textcolor{red}{\textless double precision\textgreater} x\argin)
Return the log-prior for this parameter.

double precision map(\textcolor{red}{\textless double precision\textgreater} x\argin) Map the parameter value.

type(varying_string) name() Return the name of this parameter.

type(varying_string) objectType() Return the type of the object.

double precision priorInvert(\textcolor{red}{\textless double precision\textgreater} f\argin)
Invert the prior, returning the parameter value given the cumulative probability.

double precision priorMaximum() Return the maximum non-zero value of the prior for this parameter.

double precision priorMinimum() Return the minimum non-zero value of the prior for this parameter.

double precision priorSample() Sample from the parameter's prior.

double precision randomPerturbation() Return a random perturbation for this parameter.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision unmap(\textcolor{red}{\textless double precision\textgreater} x\argin)
Unmap the parameter value.

modelParameterClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((modelParameterClass))\textgreater} destination\arginout)
Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision logPrior(\textcolor{red}{\textless double precision\textgreater} x\argin)` Return the log-prior for this parameter.

`double precision map(\textcolor{red}{\textless double precision\textgreater} x\argin)` Map the parameter value.

`type(varying_string) name()` Return the name of this parameter.

`type(varying_string) objectType()` Return the type of the object.

`double precision priorInvert(\textcolor{red}{\textless double precision\textgreater} f\argin)` Invert the prior, returning the parameter value given the cumulative probability.

`double precision priorMaximum()` Return the maximum non-zero value of the prior for this parameter.

`double precision priorMinimum()` Return the minimum non-zero value of the prior for this parameter.

`double precision priorSample()` Sample from the parameter's prior.

`double precision randomPerturbation()` Return a random perturbation for this parameter.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision unmap(\textcolor{red}{\textless double precision\textgreater} x\argin)` Unmap the parameter value.

modelParameterDerived

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((modelParameterClass))\textgreater} destination\arginout)
Perform a deep copy of the object.

<type(varying_string)> definition() Return the definition for this parameter.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision logPrior(\textcolor{red}{\textless double precision\textgreater} x\argin)
Return the log-prior for this parameter.

double precision map(\textcolor{red}{\textless double precision\textgreater} x\argin) Map the parameter value.

type(varying_string) name() Return the name of this parameter.

type(varying_string) objectType() Return the type of the object.

double precision priorInvert(\textcolor{red}{\textless double precision\textgreater} f\argin)
Invert the prior, returning the parameter value given the cumulative probability.

double precision priorMaximum() Return the maximum non-zero value of the prior for this parameter.

double precision priorMinimum() Return the minimum non-zero value of the prior for this parameter.

double precision priorSample() Sample from the parameter's prior.

double precision randomPerturbation() Return a random perturbation for this parameter.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision unmap(\textcolor{red}{\textless double precision\textgreater} x\argin)` Unmap the parameter value.

modelParameterInactive

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((modelParameterClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision logPrior(\textcolor{red}{\textless double precision\textgreater} x\argin)` Return the log-prior for this parameter.

`double precision map(\textcolor{red}{\textless double precision\textgreater} x\argin)` Map the parameter value.

`type(varying_string) name()` Return the name of this parameter.

`type(varying_string) objectType()` Return the type of the object.

`double precision priorInvert(\textcolor{red}{\textless double precision\textgreater} f\argin)` Invert the prior, returning the parameter value given the cumulative probability.

`double precision priorMaximum()` Return the maximum non-zero value of the prior for this parameter.

`double precision priorMinimum()` Return the minimum non-zero value of the prior for this parameter.

`double precision priorSample()` Sample from the parameter's prior.

`double precision randomPerturbation()` Return a random perturbation for this parameter.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision unmap(\textcolor{red}{\textless double precision\textgreater} x\argin)` Unmap the parameter value.

mpiCounter

`<integer(c_size_t)> get()` Get the current value of the counter.

`<integer(c_size_t)> increment()` Increment the counter and return the new value.

mpiObject

`<logical> all(<logical> scalar→, <logical> [mask]→)` Return true if every `scalar` is true over all processes.

`<logical> any(<logical> scalar→, <logical> [mask]→)` Return true if any of `scalar` is true over all processes.

`<double(:)> average(<double(:)>array→)` Return the average of `array` over all processes.

`<integer> count()` Return the total number of processes.

`(<double(:)>|<double(:,:)>|<double(:,:,:)>|<integer(:,:)>) gather((<double>|<double(:)>|<double(:,:)>|< array→)` Gather arrays from all processes into an array of rank one higher.

`<type(varying_string)> hostAffinity()` Return the name of the host on which this MPI process is running.

`<logical> isActive()` Return true if MPI is active.

`<logical> isMaster()` Return true if this is the master process (i.e. rank-0 process).

`<integer(:)> maxloc(<double(:)>array→)` Return the rank of the process with the maximum value of `array` over all processes.

`<double(:)> maxval((<double>|<double(:)>) array→)` Return the maximum value of `array` over all processes.

`<integer(:)> median(<integer(:)>array→)` Return the median of `array` over all processes.

`<logical> messageWaiting(<integer> [from]→, <integer> [tag]→)` Return true if a message is waiting, optionally from the specified process and with the specified tag.

`<integer(:)> minloc(<double(:)>array→)` Return the rank of the process with the minimum value of `array` over all processes.

`<double>|<double(:)>|<integer>|<integer(:)> minval((<double>|<double(:)>|<integer>|<integer(:)>) array→)` Return the minimum value of `array` over all processes.

<integer> nodeAffinity(<integer> [rank]→) Return the index of the node on which the MPI process of the given rank (or this process if no rank is given) is running.

<integer> nodeCount() Return the number of nodes on which this MPI job is running.

<integer> rank() Return the rank of this process.

<type(varying_string)> rankLabel() Return a label containing the rank of the process.

<double(:,:)>|<integer(:,:)>|<logical(:,:)> requestData(<integer(:)>requestFrom→, <double(:)>|<integer
Request the content of array from each processes listed in requestFrom.

<integer>|<integer(:)> sum((<integer>|<integer(:)>|<double>|<double(:)>|<double(:,:)>) array→) Return the sum of array over all processes.

multiCounter

<void> append(<integer> range→) Append a new counter to the multi-counter, with the specified range.

<integer> count() Return the number of counters configured in this multi-counter.

<logical> increment() Increment the state of the multi-counter. Return .false. if incrementing was not possible (i.e. counter was in the final state), .true. otherwise.

<logical> isFinal() Return .true. if the counter is in its final state, .false. otherwise.

<void> reset() Reset the multi-counter back to its initial count state, such that the next increment will return the first count.

<integer> state(<integer> i→) Return the state of the i^{th} counter.

nbodyHaloMassErrorClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

double precision correlation(\textcolor{red}{\textless type((treeNode))\textgreater} node1\arginout,\te
type((treeNode))\textgreater} node2\arginout) Return the correlation in the error on the mass of a pair of N-body halos corresponding to the given node1 and node2.

void deepCopy(\textcolor{red}{\textless class((nbodyHaloMassErrorClass))\textgreater} destination\argin
Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\te
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision errorFractional(\textcolor{red}{\textless type((treeNode))\textgreater}
node\arginout) Return the fractional error on the mass of an N-body halo corresponding to the given node.

logical errorZeroAlways() Return true if the mass error is always zero for any halo.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

nbodyHaloMassErrorFriendsOfFriends

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

double precision correlation(\textcolor{red}{\textless type((treeNode))\textgreater} node1\arginout,\te
type((treeNode))\textgreater} node2\arginout) Return the correlation in the error on the mass of a pair of N-body halos corresponding to the given node1 and node2.

void deepCopy(\textcolor{red}{\textless class((nbodyHaloMassErrorClass))\textgreater} destination\argin
Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision errorFractional(\textcolor{red}{\textless type((treeNode))\textgreater}
node\arginout) Return the fractional error on the mass of an N-body halo corresponding to the given node.

logical errorZeroAlways() Return true if the mass error is always zero for any halo.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

nbodyHaloMassErrorNull

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

double precision correlation(\textcolor{red}{\textless type((treeNode))\textgreater} node1\arginout,\te type((treeNode))\textgreater} node2\arginout) Return the correlation in the error on the mass of a pair of N-body halos corresponding to the given node1 and node2.

void deepCopy(\textcolor{red}{\textless class((nbodyHaloMassErrorClass))\textgreater} destination\argin Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision errorFractional(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Return the fractional error on the mass of an N-body halo corresponding to the given node.

logical errorZeroAlways() Return true if the mass error is always zero for any halo.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

nbodyHaloMassErrorPowerLaw

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`double precision correlation(\textcolor{red}{\textless type((treeNode))\textgreater} node1\arginout,\te type((treeNode))\textgreater} node2\arginout)` Return the correlation in the error on the mass of a pair of N-body halos corresponding to the given `node1` and `node2`.

`void deepCopy(\textcolor{red}{\textless class((nbodyHaloMassErrorClass))\textgreater} destination\argin` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision errorFractional(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Return the fractional error on the mass of an N-body halo corresponding to the given `node`.

`logical errorZeroAlways()` Return `true` if the mass error is always zero for any halo.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

nbodyHaloMassErrorSOHaloFinder

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

double precision correlation(\textcolor{red}{\textless type((treeNode))\textgreater} node1\arginout,\te type((treeNode))\textgreater} node2\arginout) Return the correlation in the error on the mass of a pair of N-body halos corresponding to the given node1 and node2.

void deepCopy(\textcolor{red}{\textless class((nbodyHaloMassErrorClass))\textgreater} destination\argin Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision errorFractional(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Return the fractional error on the mass of an N-body halo corresponding to the given node.

logical errorZeroAlways() Return true if the mass error is always zero for any halo.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

nbodyHaloMassErrorTrenti2010

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

double precision correlation(\textcolor{red}{\textless type((treeNode))\textgreater} node1\arginout,\te type((treeNode))\textgreater} node2\arginout) Return the correlation in the error on the mass of a pair of N-body halos corresponding to the given node1 and node2.

void deepCopy(\textcolor{red}{\textless class((nbodyHaloMassErrorClass))\textgreater} destination\argin Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision errorFractional(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Return the fractional error on the mass of an N-body halo corresponding to the given `node`.

`logical errorZeroAlways()` Return `true` if the mass error is always zero for any halo.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

nbodyImporterClass

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((nbodyImporterClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`type(nBodyData) import(\textcolor{red}{\textless character((len=*))\textgreater} fileName\argin,\textcc character((len=*))\textgreater} fileNamePrevious\argin)` Import position and velocity data from the named N-body data file.

`<logical> isDefault()` Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

nbodyImporterGadgetBinary

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((nbodyImporterClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

type(nBodyData) import(\textcolor{red}{\textless character((len=*))\textgreater} fileName\argin,\textcc character((len=*))\textgreater} fileNamePrevious\argin) Import position and velocity data from the named N-body data file.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

nbodyImporterGadgetHDF5

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((nbodyImporterClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`type(nBodyData) import(\textcolor{red}{\textless character((len=*))\textgreater} fileName\argin,\textco character((len=*))\textgreater} fileNamePrevious\argin)` Import position and velocity data from the named N-body data file.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

nbodyOperatorClass

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((nbodyOperatorClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

void operate(\textcolor{red}{\textless type((nBodyData))\textgreater} simulation\arginout)
Operate on the provided N-body simulation.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

nbodyOperatorEnvironmentalOverdensity

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((nbodyOperatorClass))\textgreater} destination\arginout)
Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

void operate(\textcolor{red}{\textless type((nBodyData))\textgreater} simulation\arginout)
Operate on the provided N-body simulation.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

nbodyOperatorMeanPosition

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((nbodyOperatorClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`void operate(\textcolor{red}{\textless type((nBodyData))\textgreater} simulation\arginout)` Operate on the provided N-body simulation.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

nbodyOperatorNull

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((nbodyOperatorClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`void operate(\textcolor{red}{\textless type((nBodyData))\textgreater} simulation\arginout)` Operate on the provided N-body simulation.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

nbodyOperatorPairCounts

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((nbodyOperatorClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`void operate(\textcolor{red}{\textless type((nBodyData))\textgreater} simulation\arginout)` Operate on the provided N-body simulation.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

nbodyOperatorRotationCurve

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((nbodyOperatorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

void operate(\textcolor{red}{\textless type((nBodyData))\textgreater} simulation\arginout) Operate on the provided N-body simulation.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

nbodyOperatorSelfBound

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((nbodyOperatorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

void operate(\textcolor{red}{\textless type((nBodyData))\textgreater} simulation\arginout) Operate on the provided N-body simulation.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

nbodyOperatorSequence

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((nbodyOperatorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

void operate(\textcolor{red}{\textless type((nBodyData))\textgreater} simulation\arginout)
Operate on the provided N-body simulation.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

nbodyOperatorVelocityDispersion

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((nbodyOperatorClass))\textgreater} destination\arginout)
Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

void operate(\textcolor{red}{\textless type((nBodyData))\textgreater} simulation\arginout)
Operate on the provided N-body simulation.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

nearestNeighbors

<void> search(<double(:)> point→, <integer> neighborCount→, <double> tolerance→, <integer(:)> neighborIndex←, <double(:)> neighborDistance←) Compute indices and distances to the approximate nearest neighbors.

<void> searchFixedRadius(<double(:)> point→, <double> radius→, <integer> neighborCount→, <double> tolerance→, <integer(:)> neighborIndex←, <double(:)> neighborDistance←) Compute indices and distances to the approximate nearest neighbors.

nodeComponent

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Read properties from raw file.

<void> deserializeValues(<double(:)> array→, <integer> propertyType→) Deserialize the evolvable quantities from an array.

<void> destroy() Destroy the object.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a nodeComponent object.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(:,:)> integerBuffer↔, <integer>doubleProperty↔, <integer> doubleBufferCount↔, <double(:,:)> doubleBuffer↔, <double> time→, <integer> instance→) Generate values of outputtable properties.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double> time→, <integer> instance→) Compute a count of outputtable properties.

<void> outputNames(<integer> integerProperty↔, <char[*](:)> integerPropertyNames↔, <char[*](:)> integerPropertyComments↔, <double(:)> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <char[*](:)> doublePropertyNames↔, <char[*](:)> doublePropertyComments↔, <double(:)> doublePropertyUnitsSI↔, <double> time→, <integer> instance→) Generate names of outputtable properties.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets() Set offsets into serialization arrays.

<void> serializeASCII() Generate an ASCII dump of all properties.

<integer> serializeCount(<integer> propertyType→) Return a count of the number of evolvable quantities to be evolved.

<void> serializeRaw(<integer> fileHandle→) Generate a binary dump of all properties.

<void> serializeValues(<double(:)> array←, <integer> propertyType→) Serialize the evolvable quantities to an array.

<void> serializeXML() Generate an XML dump of all properties.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<type(varying_string)> type() Return the type of this object.

nodeComponentAgeStatistics

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a generic ageStatistics component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Read properties from raw file.

<void> deserializeValues(<double(:)> array→, <integer> propertyType→) Deserialize the evolvable quantities from an array.

<void> destroy() Finalize a generic ageStatistics component.

<double precision> diskIntegratedSFR() Returns the default value for the diskIntegratedSFR property for the ageStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> diskIntegratedSFRAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the ageStatistics class that have the desired attributes for the diskIntegratedSFR property

<integer> diskIntegratedSFRCount() Compute the count of evolvable quantities in the diskIntegratedSFR property of the AgeStatisticsStandard component.

<logical> diskIntegratedSFRIsGettable() Returns true if the diskIntegratedSFR property is gettable for the ageStatistics component class.

<logical> diskIntegratedSFRIsSettable() Specify whether the diskIntegratedSFR property of the ageStatistics component is settable.

<void> diskIntegratedSFRRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interr [interruptProcedure]↔) Accept a rate set for the diskIntegratedSFR property of the ageStatistics component class. Trigger an interrupt to create the component.

<double precision> diskIntegratedSFRRateGet() Returns a zero rate for the diskIntegratedSFR property for the ageStatistics component class.

<void> diskIntegratedSFRScale(<double precision> value) Set the scale of the diskIntegratedSFR property of the AgeStatisticsStandard component.

<void> diskIntegratedSFRSet(<double precision> value) Set the diskIntegratedSFR property of the ageStatistics component.

<double precision> diskTimeWeightedIntegratedSFR() Returns the default value for the diskTimeWeightedIntegrate property for the ageStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> diskTimeWeightedIntegratedSFRAttributeMat [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the ageStatistics class that have the desired attributes for the diskTimeWeightedIntegratedSFR property

<integer> diskTimeWeightedIntegratedSFRCount() Compute the count of evolvable quantities in the diskTimeWeightedIntegratedSFR property of the AgeStatisticsStandard component.

<logical> diskTimeWeightedIntegratedSFRIsGettable() Returns true if the diskTimeWeightedIntegratedSFR property is gettable for the ageStatistics component class.

<logical> diskTimeWeightedIntegratedSFRIsSettable() Specify whether the diskTimeWeightedIntegratedSFR property of the ageStatistics component is settable.

<void> diskTimeWeightedIntegratedSFRRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accept a rate set for the diskTimeWeightedIntegratedSF property of the ageStatistics component class. Trigger an interrupt to create the component.

<double precision> diskTimeWeightedIntegratedSFRRateGet() Returns a zero rate for the diskTimeWeightedIntegra property for the ageStatistics component class.

<void> diskTimeWeightedIntegratedSFRScale(<double precision> value) Set the scale of the diskTimeWeightedInte property of the AgeStatisticsStandard component.

<void> diskTimeWeightedIntegratedSFRSet(<double precision> value) Set the diskTimeWeightedIntegratedSFR property of the ageStatistics component.

<void> dumpASCII() Dump the content of a ageStatistics component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a generic ageStatistics component.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a nodeComponent object.

<logical> nullIsActive() Return true if the null implementation of the ageStatistics component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a ageStatistics component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a generic ageStatistics component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Establish the names of properties to output for a generic ageStatistics component.

<void> postOutput(<double precision> time→) Perform post-output processing of a ageStatistics component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets() Set offsets into serialization arrays.

<void> serializeASCII() Serialize the content of a ageStatistics component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the number of evolvable quantities to be evolved.

<void> serializeRaw(<integer> fileHandle→) Generate a binary dump of all properties.

<void> serializeValues(<double(:)> array←, <integer> propertyType→) Serialize the evolvable quantities to an array.

<void> serializeXML() Generate an XML dump of all properties.

<integer(c_size_t)> sizeOf() Return the size in bytes of a nodeComponentAgeStatistics component.

<double precision> spheroidIntegratedSFR() Returns the default value for the spheroidIntegratedSFR property for the ageStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> spheroidIntegratedSFRAttributeMatch(<logi [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the ageStatistics class that have the desired attributes for the spheroidIntegratedSFR property

<integer> spheroidIntegratedSFRCount() Compute the count of evolvable quantities in the spheroidIntegratedSFR property of the AgeStatisticsStandard component.

<logical> spheroidIntegratedSFRIsGettable() Returns true if the spheroidIntegratedSFR property is gettable for the ageStatistics component class.

<logical> spheroidIntegratedSFRIsSettable() Specify whether the spheroidIntegratedSFR property of the ageStatistics component is settable.

<void> spheroidIntegratedSFRRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accept a rate set for the spheroidIntegratedSFR property of the ageStatistics component class. Trigger an interrupt to create the component.

<double precision> spheroidIntegratedSFRRateGet() Returns a zero rate for the spheroidIntegratedSFR property for the ageStatistics component class.

<void> spheroidIntegratedSFRScale(<double precision> value) Set the scale of the spheroidIntegratedSFR property of the AgeStatisticsStandard component.

<void> spheroidIntegratedSFRSet(<double precision> value) Set the spheroidIntegratedSFR property of the ageStatistics component.

<double precision> spheroidTimeWeightedIntegratedSFR() Returns the default value for the spheroidTimeWeightedI property for the ageStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> spheroidTimeWeightedIntegratedSFRAttribut [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the ageStatistics class that have the desired attributes for the spheroidTimeWeightedIntegratedSFR property

<integer> spheroidTimeWeightedIntegratedSFRCount() Compute the count of evolvable quantities in the spheroidTimeWeightedIntegratedSFR property of the AgeStatisticsStandard component.

<logical> spheroidTimeWeightedIntegratedSFRIsGettable() Returns true if the spheroidTimeWeightedIntegratedSF property is gettable for the ageStatistics component class.

<logical> spheroidTimeWeightedIntegratedSFRIsSettable() Specify whether the spheroidTimeWeightedIntegrated property of the ageStatistics component is settable.

<void> spheroidTimeWeightedIntegratedSFRRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accept a rate set for the spheroidTimeWeightedIntegrat property of the ageStatistics component class. Trigger an interrupt to create the component.

<double precision> spheroidTimeWeightedIntegratedSFRRateGet() Returns a zero rate for the spheroidTimeWeightedIntegratedSFR property for the ageStatistics component class.

<void> spheroidTimeWeightedIntegratedSFRScale(<double precision> value) Set the scale of the spheroidTimeWeightedIntegratedSFR property of the AgeStatisticsStandard component.

<void> spheroidTimeWeightedIntegratedSFRSet(<double precision> value) Set the spheroidTimeWeightedIntegra property of the ageStatistics component.

<logical> standardIsActive() Return true if the standard implementation of the ageStatistics component is the active choice.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<type(varying_string)> type() Returns the type name for the ageStatistics component class.

nodeComponentAgeStatisticsNull

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a null implementation of the ageStatistics component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a null implementation of the ageStatistics component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a null implementation of the ageStatistics component from array.

<void> destroy() Finalize a null implementation of the ageStatistics component.

<double precision> diskIntegratedSFR() Returns the default value for the diskIntegratedSFR property for the ageStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> diskIntegratedSFRAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the ageStatistics class that have the desired attributes for the diskIntegratedSFR property

<integer> diskIntegratedSFRCount() Compute the count of evolvable quantities in the diskIntegratedSFR property of the AgeStatisticsStandard component.

<logical> diskIntegratedSFRIsGettable() Returns true if the diskIntegratedSFR property is gettable for the ageStatistics component class.

<logical> diskIntegratedSFRIsSettable() Specify whether the diskIntegratedSFR property of the ageStatistics component is settable.

<void> diskIntegratedSFRRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interr [interruptProcedure]↔) Accept a rate set for the diskIntegratedSFR property of the ageStatistics component class. Trigger an interrupt to create the component.

<double precision> diskIntegratedSFRRateGet() Returns a zero rate for the diskIntegratedSFR property for the ageStatistics component class.

<void> diskIntegratedSFRScale(<double precision> value) Set the scale of the diskIntegratedSFR property of the AgeStatisticsStandard component.

<void> diskIntegratedSFRSet(<double precision> value) Set the diskIntegratedSFR property of the ageStatistics component.

<double precision> diskTimeWeightedIntegratedSFR() Returns the default value for the diskTimeWeightedIntegrate property for the ageStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> diskTimeWeightedIntegratedSFRAttributeMat [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the ageStatistics class that have the desired attributes for the diskTimeWeightedIntegratedSFR property

<integer> diskTimeWeightedIntegratedSFRCount() Compute the count of evolvable quantities in the diskTimeWeightedIntegratedSFR property of the AgeStatisticsStandard component.

<logical> diskTimeWeightedIntegratedSFRIsGettable() Returns true if the diskTimeWeightedIntegratedSFR property is gettable for the ageStatistics component class.

<logical> diskTimeWeightedIntegratedSFRIsSettable() Specify whether the diskTimeWeightedIntegratedSFR property of the ageStatistics component is settable.

<void> diskTimeWeightedIntegratedSFRRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accept a rate set for the diskTimeWeightedIntegratedSF property of the ageStatistics component class. Trigger an interrupt to create the component.

<double precision> diskTimeWeightedIntegratedSFRRateGet() Returns a zero rate for the diskTimeWeightedIntegra property for the ageStatistics component class.

<void> diskTimeWeightedIntegratedSFRScale(<double precision> value) Set the scale of the diskTimeWeightedInte property of the AgeStatisticsStandard component.

<void> diskTimeWeightedIntegratedSFRSet(<double precision> value) Set the diskTimeWeightedIntegratedSFR property of the ageStatistics component.

<void> dumpASCII() Dump the content of a ageStatistics component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a null member of the ageStatistics component.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a null implementation of the ageStatistics component class.

<logical> nullIsActive() Return true if the null implementation of the ageStatistics component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a ageStatistics component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a null implementation of the ageStatistics component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a null implementation of the ageStatistics component.

<void> postOutput(<double precision> time→) Perform post-output processing of a ageStatistics component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a null implementation of the ageStatistics component.

<void> serializeASCII() Serialize the contents of a null implementation of the ageStatistics component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a null implementation of the ageStatistics component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a null implementation of the ageStatistics component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a null implementation of the ageStatistics component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a null implementation of the ageStatistics component to XML.

<integer(c_size_t)> sizeOf() Return the size in bytes of a null implementation of the ageStatistics component.

<double precision> spheroidIntegratedSFR() Returns the default value for the spheroidIntegratedSFR property for the ageStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> spheroidIntegratedSFRAttributeMatch(<logi [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the ageStatistics class that have the desired attributes for the spheroidIntegratedSFR property

<integer> spheroidIntegratedSFRCount() Compute the count of evolvable quantities in the spheroidIntegratedSFR property of the AgeStatisticsStandard component.

<logical> spheroidIntegratedSFRIsGettable() Returns true if the spheroidIntegratedSFR property is gettable for the ageStatistics component class.

<logical> spheroidIntegratedSFRIsSettable() Specify whether the spheroidIntegratedSFR property of the ageStatistics component is settable.

<void> spheroidIntegratedSFRRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accept a rate set for the spheroidIntegratedSFR property of the ageStatistics component class. Trigger an interrupt to create the component.

<double precision> spheroidIntegratedSFRRateGet() Returns a zero rate for the spheroidIntegratedSFR property for the ageStatistics component class.

<void> spheroidIntegratedSFRScale(<double precision> value) Set the scale of the spheroidIntegratedSFR property of the AgeStatisticsStandard component.

<void> spheroidIntegratedSFRSet(<double precision> value) Set the spheroidIntegratedSFR property of the ageStatistics component.

<double precision> spheroidTimeWeightedIntegratedSFR() Returns the default value for the spheroidTimeWeightedI property for the ageStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> spheroidTimeWeightedIntegratedSFRAttribut [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the ageStatistics class that have the desired attributes for the spheroidTimeWeightedIntegratedSFR property

<integer> spheroidTimeWeightedIntegratedSFRCount() Compute the count of evolvable quantities in the spheroidTimeWeightedIntegratedSFR property of the AgeStatisticsStandard component.

<logical> spheroidTimeWeightedIntegratedSFRIsGettable() Returns true if the spheroidTimeWeightedIntegratedSF property is gettable for the ageStatistics component class.

<logical> spheroidTimeWeightedIntegratedSFRIsSettable() Specify whether the spheroidTimeWeightedIntegrated property of the ageStatistics component is settable.

<void> spheroidTimeWeightedIntegratedSFRRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accept a rate set for the spheroidTimeWeightedIntegrat property of the ageStatistics component class. Trigger an interrupt to create the component.

<double precision> spheroidTimeWeightedIntegratedSFRRateGet() Returns a zero rate for the spheroidTimeWeightedIntegratedSFR property for the ageStatistics component class.

<void> spheroidTimeWeightedIntegratedSFRScale(<double precision> value) Set the scale of the spheroidTimeWeightedIntegratedSFR property of the AgeStatisticsStandard component.

<void> spheroidTimeWeightedIntegratedSFRSet(<double precision> value) Set the spheroidTimeWeightedIntegra property of the ageStatistics component.

<logical> standardIsActive() Return true if the standard implementation of the ageStatistics component is the active choice.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<type(varying_string)> type() Returns the type name for the null implementation of the ageStatistics component class.

nodeComponentAgeStatisticsStandard

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a standard implementation of the ageStatistics component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a standard implementation of the ageStatistics component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a standard implementation of the ageStatistics component from array.

<void> destroy() Finalize a standard implementation of the ageStatistics component.

<double precision> diskIntegratedSFR() Get the diskIntegratedSFR property of an standard implementation of the ageStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> diskIntegratedSFRAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the ageStatistics class that have the desired attributes for the diskIntegratedSFR property

<integer> diskIntegratedSFRCount() Return a count of the number of scalar properties in the diskIntegratedSFR property of an standard implementation of the ageStatistics component class.

<void> diskIntegratedSFRInactive() Indicate that the diskIntegratedSFR property of an standard implementation of the ageStatistics component class is inactive for differential equation solving.

<logical> diskIntegratedSFRIsGettable() Returns true if the diskIntegratedSFR property is gettable for the ageStatistics component class.

<logical> diskIntegratedSFRIsSettable() Specify whether the diskIntegratedSFR property of the ageStatistics component is settable.

<void> diskIntegratedSFRRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interr [interruptProcedure]↔) Accumulate to the rate of change of the diskIntegratedSFR property of an standard implementation of the ageStatistics component class.

<double precision> diskIntegratedSFRRateGet() Get the rate of change of the diskIntegratedSFR property of an standard implementation of the ageStatistics component class.

<void> diskIntegratedSFRScale(<double precision> setValue→) Set the absolute scale of the diskIntegratedSFR property of an standard implementation of the ageStatistics component class.

<void> diskIntegratedSFRSet(<double precision> setValue→) Set the diskIntegratedSFR property of an standard implementation of the ageStatistics component class.

<double precision> diskTimeWeightedIntegratedSFR() Get the diskTimeWeightedIntegratedSFR property of an standard implementation of the ageStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> diskTimeWeightedIntegratedSFRAttributeMat [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the ageStatistics class that have the desired attributes for the diskTimeWeightedIntegratedSFR property

<integer> diskTimeWeightedIntegratedSFRCount() Return a count of the number of scalar properties in the diskTimeWeightedIntegratedSFR property of an standard implementation of the ageStatistics component class.

<void> diskTimeWeightedIntegratedSFRInactive() Indicate that the diskTimeWeightedIntegratedSFR property of an standard implementation of the ageStatistics component class is inactive for differential equation solving.

<logical> diskTimeWeightedIntegratedSFRIsGettable() Returns true if the diskTimeWeightedIntegratedSFR property is gettable for the ageStatistics component class.

<logical> diskTimeWeightedIntegratedSFRIsSettable() Specify whether the diskTimeWeightedIntegratedSFR property of the ageStatistics component is settable.

<void> diskTimeWeightedIntegratedSFRRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the diskTimeWeightedIntegratedSFR property of an standard implementation of the ageStatistics component class.

<double precision> diskTimeWeightedIntegratedSFRRateGet() Get the rate of change of the diskTimeWeightedIntegr property of an standard implementation of the ageStatistics component class.

<void> diskTimeWeightedIntegratedSFRScale(<double precision> setValue→) Set the absolute scale of the diskTimeWeightedIntegratedSFR property of an standard implementation of the ageStatistics component class.

<void> diskTimeWeightedIntegratedSFRSet(<double precision> setValue→) Set the diskTimeWeightedIntegrated property of an standard implementation of the ageStatistics component class.

<void> dumpASCII() Dump the content of a ageStatistics component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter
[interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a standard member of the ageStatistics component.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a standard implementation of the ageStatistics component class.

<logical> nullIsActive() Return true if the null implementation of the ageStatistics component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_-
int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔,
<double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)>
outputInstance→, <integer> instance→) Populate output buffers with properties to output for a ageStatistics component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double
precision> time→, <integer> instance→) Increment the count of properties to output for a standard implementation of the ageStatistics component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔,
<character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔,
<integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]>
doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision>
time→, <integer> instance→) Return the names of properties to output for a standard implementation of the ageStatistics component.

<void> postOutput(<double precision> time→) Perform post-output processing of a ageStatistics component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→)
Compute the gravitational potential.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType>
[massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType>
[massType]→) Compute the rotation curve gradient.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→)
Compute offsets into serialization arrays for a standard implementation of the ageStatistics component.

<void> serializeASCII() Serialize the contents of a standard implementation of the ageStatistics component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a standard implementation of the ageStatistics component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a standard implementation of the ageStatistics component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a standard implementation of the ageStatistics component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a standard implementation of the ageStatistics component to XML.

<integer(c_size_t)> sizeOf() Return the size in bytes of a standard implementation of the ageStatistics component.

<double precision> spheroidIntegratedSFR() Get the spheroidIntegratedSFR property of an standard implementation of the ageStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> spheroidIntegratedSFRAttributeMatch(<logi
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the ageStatistics class that have the desired attributes for the spheroidIntegratedSFR property

<integer> spheroidIntegratedSFRCount() Return a count of the number of scalar properties in the spheroidIntegratedSFR property of an standard implementation of the ageStatistics component class.

<void> spheroidIntegratedSFRInactive() Indicate that the spheroidIntegratedSFR property of an standard implementation of the ageStatistics component class is inactive for differential equation solving.

<logical> spheroidIntegratedSFRIsGettable() Returns true if the spheroidIntegratedSFR property is gettable for the ageStatistics component class.

<logical> spheroidIntegratedSFRIsSettable() Specify whether the spheroidIntegratedSFR property of the ageStatistics component is settable.

<void> spheroidIntegratedSFRRate(<double precision> setValue→, <logical> [interrupt]↔,
<*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the spheroidIntegratedSFR property of an standard implementation of the ageStatistics component class.

<double precision> spheroidIntegratedSFRRateGet() Get the rate of change of the spheroidIntegratedSFR property of an standard implementation of the ageStatistics component class.

<void> spheroidIntegratedSFRScale(<double precision> setValue→) Set the absolute scale of the spheroidIntegratedSFR property of an standard implementation of the ageStatistics component class.

<void> spheroidIntegratedSFRSet(<double precision> setValue→) Set the spheroidIntegratedSFR property of an standard implementation of the ageStatistics component class.

<double precision> spheroidTimeWeightedIntegratedSFR() Get the spheroidTimeWeightedIntegratedSFR property of an standard implementation of the ageStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> spheroidTimeWeightedIntegratedSFRAttribut
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the ageStatistics class that have the desired attributes for the spheroidTimeWeightedIntegratedSFR property

<integer> spheroidTimeWeightedIntegratedSFRCount() Return a count of the number of scalar properties in the spheroidTimeWeightedIntegratedSFR property of an standard implementation of the ageStatistics component class.

<void> spheroidTimeWeightedIntegratedSFRInactive() Indicate that the spheroidTimeWeightedIntegratedSFR property of an standard implementation of the ageStatistics component class is inactive for differential equation solving.

<logical> spheroidTimeWeightedIntegratedSFRIsGettable() Returns true if the spheroidTimeWeightedIntegratedSF property is gettable for the ageStatistics component class.

<logical> spheroidTimeWeightedIntegratedSFRIsSettable() Specify whether the spheroidTimeWeightedIntegrated property of the ageStatistics component is settable.

<void> spheroidTimeWeightedIntegratedSFRRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the spheroidTimeWeightedIntegratedSFR property of an standard implementation of the ageStatistics component class.

<double precision> spheroidTimeWeightedIntegratedSFRRateGet() Get the rate of change of the spheroidTimeWeightedIntegratedSFR property of an standard implementation of the ageStatistics component class.

<void> spheroidTimeWeightedIntegratedSFRScale(<double precision> setValue→) Set the absolute scale of the spheroidTimeWeightedIntegratedSFR property of an standard implementation of the ageStatistics component class.

<void> spheroidTimeWeightedIntegratedSFRSet(<double precision> setValue→) Set the spheroidTimeWeightedIn property of an standard implementation of the ageStatistics component class.

<logical> standardIsActive() Return true if the standard implementation of the ageStatistics component is the active choice.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<type(varying_string)> type() Returns the type name for the standard implementation of the ageStatistics component class.

nodeComponentBasic

<double precision> accretionRate() Returns the default value for the accretionRate property for the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> accretionRateAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the accretionRate property

<double precision> accretionRateBertschinger() Returns the default value for the accretionRateBertschinger property for the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> accretionRateBertschingerAttributeMatch([requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the accretionRateBertschinger property

<logical> accretionRateBertschingerIsGettable() Returns true if the accretionRateBertschinger property is gettable for the basic component class.

<logical> accretionRateBertschingerIsSettable() Specify whether the accretionRateBertschinger property of the basic component is settable.

<double precision> accretionRateBertschingerRateGet() Returns a zero rate for the accretionRateBertschinger property for the basic component class.

<void> accretionRateBertschingerSet(<double precision> value) Set the accretionRateBertschinger property of the basic component.

<logical> accretionRateIsGettable() Returns true if the accretionRate property is gettable for the basic component class.

<logical> accretionRateIsSettable() Specify whether the accretionRate property of the basic component is settable.

<double precision> accretionRateRateGet() Returns a zero rate for the accretionRate property for the basic component class.

<void> accretionRateSet(<double precision> value) Set the accretionRate property of the basic component.

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a generic basic component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Read properties from raw file.

<void> deserializeValues(<double(:)> array→, <integer> propertyType→) Deserialize the evolvable quantities from an array.

<void> destroy() Finalize a generic basic component.

<void> dumpASCII() Dump the content of a basic component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<logical> extendedTrackingIsActive() Return true if the extendedTracking implementation of the basic component is the active choice.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a generic basic component.

<double precision> mass() Returns the default value for the mass property for the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> massAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the mass property

<double precision> massBertschinger() Returns the default value for the massBertschinger property for the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> massBertschingerAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the massBertschinger property

<integer> massBertschingerCount() Compute the count of evolvable quantities in the massBertschinger property of the BasicStandardExtended component.

<logical> massBertschingerIsGettable() Returns true if the massBertschinger property is gettable for the basic component class.

<logical> massBertschingerIsSettable() Specify whether the massBertschinger property of the basic component is settable.

<void> massBertschingerRate(<double precision> value) Cumulate to the rate of the massBertschinger property of the BasicStandardExtended component.

<double precision> massBertschingerRateGet() Returns a zero rate for the massBertschinger property for the basic component class.

<void> massBertschingerScale(<double precision> value) Set the scale of the massBertschinger property of the BasicStandardExtended component.

<void> massBertschingerSet(<double precision> value) Set the massBertschinger property of the basic component.

<integer> massCount() Compute the count of evolvable quantities in the mass property of the BasicStandard component.

<logical> massIsGettable() Returns true if the mass property is gettable for the basic component class.

<logical> massIsSettable() Specify whether the mass property of the basic component is settable.

<double precision> massMaximum() Returns the default value for the massMaximum property for the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> massMaximumAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the massMaximum property

<logical> massMaximumIsGettable() Returns true if the massMaximum property is gettable for the basic component class.

<logical> massMaximumIsSettable() Specify whether the massMaximum property of the basic component is settable.

<double precision> massMaximumRateGet() Returns a zero rate for the massMaximum property for the basic component class.

<void> massMaximumSet(<double precision> value) Set the massMaximum property of the basic component.

<void> massRate(<double precision> value) Cumulate to the rate of the mass property of the BasicStandard component.

<double precision> massRateGet() Returns a zero rate for the mass property for the basic component class.

<void> massScale(<double precision> value) Set the scale of the mass property of the BasicStandard component.

<void> massSet(<double precision> value) Set the mass property of the basic component.

<double precision> massTarget() Returns the default value for the massTarget property for the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> massTargetAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the massTarget property

<logical> massTargetIsGettable() Returns true if the massTarget property is gettable for the basic component class.

<logical> massTargetIsSettable() Specify whether the massTarget property of the basic component is settable.

<double precision> massTargetRateGet() Returns a zero rate for the massTarget property for the basic component class.

<void> massTargetSet(<double precision> value) Set the massTarget property of the basic component.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a nodeComponent object.

<logical> nonEvolvingIsActive() Return true if the nonEvolving implementation of the basic component is the active choice.

<logical> nullIsActive() Return true if the null implementation of the basic component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a basic component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a generic basic component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Establish the names of properties to output for a generic basic component.

<void> postOutput(<double precision> time→) Perform post-output processing of a basic component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<double precision> radiusTurnaround() Returns the default value for the radiusTurnaround property for the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> radiusTurnaroundAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the radiusTurnaround property

<integer> radiusTurnaroundCount() Compute the count of evolvable quantities in the radiusTurnaround property of the BasicStandardExtended component.

<double precision> radiusTurnaroundGrowthRate() Returns the default value for the radiusTurnaroundGrowthRate property for the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> radiusTurnaroundGrowthRateAttributeMatch([requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the radiusTurnaroundGrowthRate property

<logical> radiusTurnaroundGrowthRateIsGettable() Returns true if the radiusTurnaroundGrowthRate property is gettable for the basic component class.

<logical> radiusTurnaroundGrowthRateIsSettable() Specify whether the radiusTurnaroundGrowthRate property of the basic component is settable.

<double precision> radiusTurnaroundGrowthRateRateGet() Returns a zero rate for the radiusTurnaroundGrowthRate property for the basic component class.

<void> radiusTurnaroundGrowthRateSet(<double precision> value) Set the radiusTurnaroundGrowthRate property of the basic component.

<logical> radiusTurnaroundIsGettable() Returns true if the radiusTurnaround property is gettable for the basic component class.

<logical> radiusTurnaroundIsSettable() Specify whether the radiusTurnaround property of the basic component is settable.

<void> radiusTurnaroundRate(<double precision> value) Cumulate to the rate of the radiusTurnaround property of the BasicStandardExtended component.

<double precision> radiusTurnaroundRateGet() Returns a zero rate for the radiusTurnaround property for the basic component class.

<void> radiusTurnaroundScale(<double precision> value) Set the scale of the radiusTurnaround property of the BasicStandardExtended component.

<void> radiusTurnaroundSet(<double precision> value) Set the radiusTurnaround property of the basic component.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets() Set offsets into serialization arrays.

<void> serializeASCII() Serialize the content of a basic component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the number of evolvable quantities to be evolved.

<void> serializeRaw(<integer> fileHandle→) Generate a binary dump of all properties.

<void> serializeValues(<double(:)> array←, <integer> propertyType→) Serialize the evolvable quantities to an array.

<void> serializeXML() Generate an XML dump of all properties.

<integer(c_size_t)> sizeOf() Return the size in bytes of a nodeComponentBasic component.

<logical> standardExtendedIsActive() Return true if the standardExtended implementation of the basic component is the active choice.

<logical> standardIsActive() Return true if the standard implementation of the basic component is the active choice.

<logical> standardTrackingIsActive() Return true if the standardTracking implementation of the basic component is the active choice.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<double precision> time() Returns the default value for the time property for the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> timeAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the time property

<integer> timeCount() Compute the count of evolvable quantities in the time property of the BasicNonEvolving component.

<logical> timeIsGettable() Returns true if the time property is gettable for the basic component class.

<logical> timeIsSettable() Specify whether the time property of the basic component is settable.

<double precision> timeLastIsolated() Returns the default value for the timeLastIsolated property for the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> timeLastIsolatedAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the timeLastIsolated property

<logical> timeLastIsolatedIsGettable() Returns true if the timeLastIsolated property is gettable for the basic component class.

<logical> timeLastIsolatedIsSettable() Specify whether the timeLastIsolated property of the basic component is settable.

<double precision> timeLastIsolatedRateGet() Returns a zero rate for the timeLastIsolated property for the basic component class.

<void> timeLastIsolatedSet(<double precision> value) Set the timeLastIsolated property of the basic component.

<void> timeRate(<double precision> value) Cumulate to the rate of the time property of the BasicNonEvolving component.

<double precision> timeRateGet() Returns a zero rate for the time property for the basic component class.

<void> timeScale(<double precision> value) Set the scale of the time property of the BasicNonEvolving component.

<void> timeSet(<double precision> value) Set the time property of the basic component.

<type(varying_string)> type() Returns the type name for the basic component class.

nodeComponentBasicExtendedTracking

<double precision> accretionRate() Get the accretionRate property of an standard implementation of the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> accretionRateAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the accretionRate property

<double precision> accretionRateBertschinger() Get the accretionRateBertschinger property of an standardExtended implementation of the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> accretionRateBertschingerAttributeMatch(< [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the accretionRateBertschinger property

<logical> accretionRateBertschingerIsGettable() Returns true if the accretionRateBertschinger property is gettable for the basic component class.

<logical> accretionRateBertschingerIsSettable() Specify whether the accretionRateBertschinger property of the basic component is settable.

<double precision> accretionRateBertschingerRateGet() Returns a zero rate for the accretionRateBertschinger property for the basic component class.

<void> accretionRateBertschingerSet(<double precision> setValue→) Set the accretionRateBertschinger property of an standardExtended implementation of the basic component class.

<logical> accretionRateIsGettable() Returns true if the accretionRate property is gettable for the basic component class.

<logical> accretionRateIsSettable() Specify whether the accretionRate property of the basic component is settable.

<double precision> accretionRateRateGet() Returns a zero rate for the accretionRate property for the basic component class.

<void> accretionRateSet(<double precision> setValue→) Set the accretionRate property of an standard implementation of the basic component class.

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a extendedTracking implementation of the basic component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a extendedTracking implementation of the basic component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a extendedTracking implementation of the basic component from array.

<void> destroy() Finalize a extendedTracking implementation of the basic component.

<void> dumpASCII() Dump the content of a basic component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<logical> extendedTrackingIsActive() Return true if the extendedTracking implementation of the basic component is the active choice.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a extendedTracking member of the basic component.

<double precision> mass() Get the mass property of an standard implementation of the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> massAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the mass property

<double precision> massBertschinger() Get the value of the massBertschinger property of the standardExtended implementation of the basic component using a deferred function.

<type(varying_string), allocatable, dimension(:) => matches> massBertschingerAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the massBertschinger property

<integer> massBertschingerCount() Return a count of the number of scalar properties in the massBertschinger property of an standardExtended implementation of the basic component class.

<void> massBertschingerFunction() Set the function to be used for the get method of the massBertschinger property of the BasicStandardExtended component.

<void> massBertschingerInactive() Indicate that the massBertschinger property of an standardExtended implementation of the basic component class is inactive for differential equation solving.

<logical> massBertschingerIsAttached() Return true if the deferred function used to get the massBertschinger property of the BasicStandardExtended component class has been attached.

<logical> massBertschingerIsGettable() Returns true if the massBertschinger property is gettable for the basic component class.

<logical> massBertschingerIsSettable() Specify whether the massBertschinger property of the basic component is settable.

<void> massBertschingerRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interru [interruptProcedure]↔) Accumulate to the rate of change of the massBertschinger property of an standardExtended implementation of the basic component class.

<double precision> massBertschingerRateGet() Get the rate of change of the massBertschinger property of an standardExtended implementation of the basic component class.

<void> massBertschingerScale(<double precision> setValue→) Set the absolute scale of the massBertschinger property of an standardExtended implementation of the basic component class.

<void> massBertschingerSet(<double precision> setValue→) Set the massBertschinger property of an standardExtended implementation of the basic component class.

<double precision> massBertschingerValue() Get the massBertschinger property of an standardExtended implementation of the basic component class.

<integer> massCount() Return a count of the number of scalar properties in the mass property of an standard implementation of the basic component class.

<void> massInactive() Indicate that the mass property of an standard implementation of the basic component class is inactive for differential equation solving.

<logical> massIsGettable() Returns true if the mass property is gettable for the basic component class.

<logical> massIsSettable() Specify whether the mass property of the basic component is settable.

<double precision> massMaximum() Get the massMaximum property of an extendedTracking implementation of the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> massMaximumAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the massMaximum property

<logical> massMaximumIsGettable() Returns true if the massMaximum property is gettable for the basic component class.

<logical> massMaximumIsSettable() Specify whether the massMaximum property of the basic component is settable.

<double precision> massMaximumRateGet() Returns a zero rate for the massMaximum property for the basic component class.

<void> massMaximumSet(<double precision> setValue→) Set the massMaximum property of an extendedTracking implementation of the basic component class.

<void> massRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the mass property of an standard implementation of the basic component class.

<double precision> massRateGet() Get the rate of change of the mass property of an standard implementation of the basic component class.

<void> massScale(<double precision> setValue→) Set the absolute scale of the mass property of an standard implementation of the basic component class.

<void> massSet(<double precision> setValue→) Set the mass property of an standard implementation of the basic component class.

<double precision> massTarget() Get the massTarget property of an standard implementation of the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> massTargetAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the massTarget property

<logical> massTargetIsGettable() Returns true if the massTarget property is gettable for the basic component class.

<logical> massTargetIsSettable() Specify whether the massTarget property of the basic component is settable.

<double precision> massTargetRateGet() Returns a zero rate for the massTarget property for the basic component class.

<void> massTargetSet(<double precision> setValue→) Set the massTarget property of an standard implementation of the basic component class.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a extendedTracking implementation of the basic component class.

<logical> nonEvolvingIsActive() Return true if the nonEvolving implementation of the basic component is the active choice.

<logical> nullIsActive() Return true if the null implementation of the basic component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a basic component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a extendedTracking implementation of the basic component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a extendedTracking implementation of the basic component.

<void> postOutput(<double precision> time→) Perform post-output processing for a extendedTracking implementation of the basic component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<double precision> radiusTurnaround() Get the value of the radiusTurnaround property of the standardExtended implementation of the basic component using a deferred function.

<type(varying_string), allocatable, dimension(:) => matches> radiusTurnaroundAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the radiusTurnaround property

<integer> radiusTurnaroundCount() Return a count of the number of scalar properties in the radiusTurnaround property of an standardExtended implementation of the basic component class.

<void> radiusTurnaroundFunction() Set the function to be used for the get method of the radiusTurnaround property of the BasicStandardExtended component.

<double precision> radiusTurnaroundGrowthRate() Get the radiusTurnaroundGrowthRate property of an standardExtended implementation of the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> radiusTurnaroundGrowthRateAttributeMatch([requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the radiusTurnaroundGrowthRate property

<logical> radiusTurnaroundGrowthRateIsGettable() Returns true if the radiusTurnaroundGrowthRate property is gettable for the basic component class.

<logical> radiusTurnaroundGrowthRateIsSettable() Specify whether the radiusTurnaroundGrowthRate property of the basic component is settable.

<double precision> radiusTurnaroundGrowthRateRateGet() Returns a zero rate for the radiusTurnaroundGrowthRate property for the basic component class.

<void> radiusTurnaroundGrowthRateSet(<double precision> setValue→) Set the radiusTurnaroundGrowthRate property of an standardExtended implementation of the basic component class.

<void> radiusTurnaroundInactive() Indicate that the radiusTurnaround property of an standardExtended implementation of the basic component class is inactive for differential equation solving.

<logical> radiusTurnaroundIsAttached() Return true if the deferred function used to get the radiusTurnaround property of the BasicStandardExtended component class has been attached.

<logical> radiusTurnaroundIsGettable() Returns true if the radiusTurnaround property is gettable for the basic component class.

<logical> radiusTurnaroundIsSettable() Specify whether the radiusTurnaround property of the basic component is settable.

<void> radiusTurnaroundRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interru [interruptProcedure]↔) Accumulate to the rate of change of the radiusTurnaround property of an standardExtended implementation of the basic component class.

<double precision> radiusTurnaroundRateGet() Get the rate of change of the radiusTurnaround property of an standardExtended implementation of the basic component class.

<void> radiusTurnaroundScale(<double precision> setValue→) Set the absolute scale of the radiusTurnaround property of an standardExtended implementation of the basic component class.

<void> radiusTurnaroundSet(<double precision> setValue→) Set the radiusTurnaround property of an standardExtended implementation of the basic component class.

<double precision> radiusTurnaroundValue() Get the radiusTurnaround property of an standardExtended implementation of the basic component class.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a extendedTracking implementation of the basic component.

<void> serializeASCII() Serialize the contents of a extendedTracking implementation of the basic component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a extendedTracking implementation of the basic component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a extendedTracking implementation of the basic component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a extendedTracking implementation of the basic component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a extendedTracking implementation of the basic component to XML.

<integer(c_size_t)> sizeOf() Return the size in bytes of a extendedTracking implementation of the basic component.

<logical> standardExtendedIsActive() Return true if the standardExtended implementation of the basic component is the active choice.

<logical> standardIsActive() Return true if the standard implementation of the basic component is the active choice.

<logical> standardTrackingIsActive() Return true if the standardTracking implementation of the basic component is the active choice.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<double precision> time() Get the time property of an standard implementation of the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> timeAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the time property

<integer> timeCount() Return a count of the number of scalar properties in the time property of an standard implementation of the basic component class.

<void> timeInactive() Indicate that the time property of an standard implementation of the basic component class is inactive for differential equation solving.

<logical> timeIsGettable() Returns true if the time property is gettable for the basic component class.

<logical> timeIsSettable() Specify whether the time property of the basic component is settable.

<double precision> timeLastIsolated() Returns the default value for the timeLastIsolated property for the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> timeLastIsolatedAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the timeLastIsolated property

<logical> timeLastIsolatedIsGettable() Returns true if the timeLastIsolated property is gettable for the basic component class.

<logical> timeLastIsolatedIsSettable() Specify whether the timeLastIsolated property of the basic component is settable.

<double precision> timeLastIsolatedRateGet() Returns a zero rate for the timeLastIsolated property for the basic component class.

<void> timeLastIsolatedSet(<double precision> setValue→) Set the timeLastIsolated property of an standard implementation of the basic component class.

<void> timeRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the time property of an standard implementation of the basic component class.

<double precision> timeRateGet() Get the rate of change of the time property of an standard implementation of the basic component class.

<void> timeScale(<double precision> setValue→) Set the absolute scale of the time property of an standard implementation of the basic component class.

<void> timeSet(<double precision> setValue→) Set the time property of an standard implementation of the basic component class.

<type(varying_string)> type() Returns the type name for the extendedTracking implementation of the basic component class.

nodeComponentBasicNonEvolving

<double precision> accretionRate() Returns the default value for the accretionRate property for the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> accretionRateAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the accretionRate property

<double precision> accretionRateBertschinger() Returns the default value for the accretionRateBertschinger property for the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> accretionRateBertschingerAttributeMatch(< [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the accretionRateBertschinger property

<logical> accretionRateBertschingerIsGettable() Returns true if the accretionRateBertschinger property is gettable for the basic component class.

<logical> accretionRateBertschingerIsSettable() Specify whether the accretionRateBertschinger property of the basic component is settable.

<double precision> accretionRateBertschingerRateGet() Returns a zero rate for the accretionRateBertschinger property for the basic component class.

<void> accretionRateBertschingerSet(<double precision> value) Set the accretionRateBertschinger property of the basic component.

<logical> accretionRateIsGettable() Returns true if the accretionRate property is gettable for the basic component class.

<logical> accretionRateIsSettable() Specify whether the accretionRate property of the basic component is settable.

<double precision> accretionRateRateGet() Returns a zero rate for the accretionRate property for the basic component class.

<void> accretionRateSet(<double precision> value) Set the accretionRate property of the basic component.

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a nonEvolving implementation of the basic component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a nonEvolving implementation of the basic component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a nonEvolving implementation of the basic component from array.

<void> destroy() Finalize a nonEvolving implementation of the basic component.

<void> dumpASCII() Dump the content of a basic component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<logical> extendedTrackingIsActive() Return true if the extendedTracking implementation of the basic component is the active choice.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a nonEvolving member of the basic component.

<double precision> mass() Get the mass property of an nonEvolving implementation of the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> massAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the mass property

<double precision> massBertschinger() Returns the default value for the massBertschinger property for the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> massBertschingerAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the massBertschinger property

<integer> massBertschingerCount() Compute the count of evolvable quantities in the massBertschinger property of the BasicStandardExtended component.

<logical> massBertschingerIsGettable() Returns true if the massBertschinger property is gettable for the basic component class.

<logical> massBertschingerIsSettable() Specify whether the massBertschinger property of the basic component is settable.

<void> massBertschingerRate(<double precision> value) Cumulate to the rate of the massBertschinger property of the BasicStandardExtended component.

<double precision> massBertschingerRateGet() Returns a zero rate for the massBertschinger property for the basic component class.

<void> massBertschingerScale(<double precision> value) Set the scale of the massBertschinger property of the BasicStandardExtended component.

<void> massBertschingerSet(<double precision> value) Set the massBertschinger property of the basic component.

<integer> massCount() Compute the count of evolvable quantities in the mass property of the BasicStandard component.

<logical> massIsGettable() Returns true if the mass property is gettable for the basic component class.

<logical> massIsSettable() Specify whether the mass property of the basic component is settable.

<double precision> massMaximum() Returns the default value for the massMaximum property for the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> massMaximumAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the massMaximum property

<logical> massMaximumIsGettable() Returns true if the massMaximum property is gettable for the basic component class.

<logical> massMaximumIsSettable() Specify whether the massMaximum property of the basic component is settable.

<double precision> massMaximumRateGet() Returns a zero rate for the massMaximum property for the basic component class.

<void> massMaximumSet(<double precision> value) Set the massMaximum property of the basic component.

<void> massRate(<double precision> value) Cumulate to the rate of the mass property of the BasicStandard component.

<double precision> massRateGet() Returns a zero rate for the mass property for the basic component class.

<void> massScale(<double precision> value) Set the scale of the mass property of the BasicStandard component.

<void> massSet(<double precision> setValue→) Set the mass property of an nonEvolving implementation of the basic component class.

<double precision> massTarget() Returns the default value for the massTarget property for the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> massTargetAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the massTarget property

<logical> massTargetIsGettable() Returns true if the massTarget property is gettable for the basic component class.

<logical> massTargetIsSettable() Specify whether the massTarget property of the basic component is settable.

<double precision> massTargetRateGet() Returns a zero rate for the massTarget property for the basic component class.

<void> massTargetSet(<double precision> value) Set the massTarget property of the basic component.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a nonEvolving implementation of the basic component class.

<logical> nonEvolvingIsActive() Return true if the nonEvolving implementation of the basic component is the active choice.

<logical> nullIsActive() Return true if the null implementation of the basic component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a basic component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a nonEvolving implementation of the basic component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a nonEvolving implementation of the basic component.

<void> postOutput(<double precision> time→) Perform post-output processing of a basic component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<double precision> radiusTurnaround() Returns the default value for the radiusTurnaround property for the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> radiusTurnaroundAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the radiusTurnaround property

<integer> radiusTurnaroundCount() Compute the count of evolvable quantities in the radiusTurnaround property of the BasicStandardExtended component.

<double precision> radiusTurnaroundGrowthRate() Returns the default value for the radiusTurnaroundGrowthRate property for the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> radiusTurnaroundGrowthRateAttributeMatch([requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the radiusTurnaroundGrowthRate property

<logical> radiusTurnaroundGrowthRateIsGettable() Returns true if the radiusTurnaroundGrowthRate property is gettable for the basic component class.

<logical> radiusTurnaroundGrowthRateIsSettable() Specify whether the radiusTurnaroundGrowthRate property of the basic component is settable.

<double precision> radiusTurnaroundGrowthRateRateGet() Returns a zero rate for the radiusTurnaroundGrowthRate property for the basic component class.

<void> radiusTurnaroundGrowthRateSet(<double precision> value) Set the radiusTurnaroundGrowthRate property of the basic component.

<logical> radiusTurnaroundIsGettable() Returns true if the radiusTurnaround property is gettable for the basic component class.

<logical> radiusTurnaroundIsSettable() Specify whether the radiusTurnaround property of the basic component is settable.

<void> radiusTurnaroundRate(<double precision> value) Cumulate to the rate of the radiusTurnaround property of the BasicStandardExtended component.

<double precision> radiusTurnaroundRateGet() Returns a zero rate for the radiusTurnaround property for the basic component class.

<void> radiusTurnaroundScale(<double precision> value) Set the scale of the radiusTurnaround property of the BasicStandardExtended component.

<void> radiusTurnaroundSet(<double precision> value) Set the radiusTurnaround property of the basic component.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a nonEvolving implementation of the basic component.

<void> serializeASCII() Serialize the contents of a nonEvolving implementation of the basic component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a nonEvolving implementation of the basic component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a nonEvolving implementation of the basic component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a nonEvolving implementation of the basic component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a nonEvolving implementation of the basic component to XML.

<integer(c_size_t)> sizeOf() Return the size in bytes of a nonEvolving implementation of the basic component.

<logical> standardExtendedIsActive() Return true if the standardExtended implementation of the basic component is the active choice.

<logical> standardIsActive() Return true if the standard implementation of the basic component is the active choice.

<logical> standardTrackingIsActive() Return true if the standardTracking implementation of the basic component is the active choice.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<double precision> time() Get the time property of an nonEvolving implementation of the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> timeAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the time property

<integer> timeCount() Return a count of the number of scalar properties in the time property of an nonEvolving implementation of the basic component class.

<void> timeInactive() Indicate that the time property of an nonEvolving implementation of the basic component class is inactive for differential equation solving.

<logical> timeIsGettable() Returns true if the time property is gettable for the basic component class.

<logical> timeIsSettable() Specify whether the time property of the basic component is settable.

<double precision> timeLastIsolated() Returns the default value for the timeLastIsolated property for the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> timeLastIsolatedAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the timeLastIsolated property

<logical> timeLastIsolatedIsGettable() Returns true if the timeLastIsolated property is gettable for the basic component class.

<logical> timeLastIsolatedIsSettable() Specify whether the timeLastIsolated property of the basic component is settable.

<double precision> timeLastIsolatedRateGet() Returns a zero rate for the timeLastIsolated property for the basic component class.

<void> timeLastIsolatedSet(<double precision> setValue→) Set the timeLastIsolated property of an nonEvolving implementation of the basic component class.

<void> timeRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the time property of an nonEvolving implementation of the basic component class.

<double precision> timeRateGet() Get the rate of change of the time property of an nonEvolving implementation of the basic component class.

<void> timeScale(<double precision> setValue→) Set the absolute scale of the time property of an nonEvolving implementation of the basic component class.

<void> timeSet(<double precision> setValue→) Set the time property of an nonEvolving implementation of the basic component class.

<type(varying_string)> type() Returns the type name for the nonEvolving implementation of the basic component class.

nodeComponentBasicNull

<double precision> accretionRate() Returns the default value for the accretionRate property for the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> accretionRateAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the accretionRate property

<double precision> accretionRateBertschinger() Returns the default value for the accretionRateBertschinger property for the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> accretionRateBertschingerAttributeMatch(< [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the accretionRateBertschinger property

<logical> accretionRateBertschingerIsGettable() Returns true if the accretionRateBertschinger property is gettable for the basic component class.

<logical> accretionRateBertschingerIsSettable() Specify whether the accretionRateBertschinger property of the basic component is settable.

<double precision> accretionRateBertschingerRateGet() Returns a zero rate for the accretionRateBertschinger property for the basic component class.

<void> accretionRateBertschingerSet(<double precision> value) Set the accretionRateBertschinger property of the basic component.

<logical> accretionRateIsGettable() Returns true if the accretionRate property is gettable for the basic component class.

<logical> accretionRateIsSettable() Specify whether the accretionRate property of the basic component is settable.

<double precision> accretionRateRateGet() Returns a zero rate for the accretionRate property for the basic component class.

<void> accretionRateSet(<double precision> value) Set the accretionRate property of the basic component.

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a null implementation of the basic component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a null implementation of the basic component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a null implementation of the basic component from array.

<void> destroy() Finalize a null implementation of the basic component.

<void> dumpASCII() Dump the content of a basic component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<logical> extendedTrackingIsActive() Return true if the extendedTracking implementation of the basic component is the active choice.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a null member of the basic component.

<double precision> mass() Returns the default value for the mass property for the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> massAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the mass property

<double precision> massBertschinger() Returns the default value for the massBertschinger property for the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> massBertschingerAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the massBertschinger property

<integer> massBertschingerCount() Compute the count of evolvable quantities in the massBertschinger property of the BasicStandardExtended component.

<logical> massBertschingerIsGettable() Returns true if the massBertschinger property is gettable for the basic component class.

<logical> massBertschingerIsSettable() Specify whether the massBertschinger property of the basic component is settable.

<void> massBertschingerRate(<double precision> value) Cumulate to the rate of the massBertschinger property of the BasicStandardExtended component.

<double precision> massBertschingerRateGet() Returns a zero rate for the massBertschinger property for the basic component class.

<void> massBertschingerScale(<double precision> value) Set the scale of the massBertschinger property of the BasicStandardExtended component.

<void> massBertschingerSet(<double precision> value) Set the massBertschinger property of the basic component.

<integer> massCount() Compute the count of evolvable quantities in the mass property of the BasicStandard component.

<logical> massIsGettable() Returns true if the mass property is gettable for the basic component class.

<logical> massIsSettable() Specify whether the mass property of the basic component is settable.

<double precision> massMaximum() Returns the default value for the massMaximum property for the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> massMaximumAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the massMaximum property

<logical> massMaximumIsGettable() Returns true if the massMaximum property is gettable for the basic component class.

<logical> massMaximumIsSettable() Specify whether the massMaximum property of the basic component is settable.

<double precision> massMaximumRateGet() Returns a zero rate for the massMaximum property for the basic component class.

<void> massMaximumSet(<double precision> value) Set the massMaximum property of the basic component.

<void> massRate(<double precision> value) Cumulate to the rate of the mass property of the BasicStandard component.

<double precision> massRateGet() Returns a zero rate for the mass property for the basic component class.

<void> massScale(<double precision> value) Set the scale of the mass property of the BasicStandard component.

<void> massSet(<double precision> value) Set the mass property of the basic component.

<double precision> massTarget() Returns the default value for the massTarget property for the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> massTargetAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the massTarget property

<logical> massTargetIsGettable() Returns true if the massTarget property is gettable for the basic component class.

<logical> massTargetIsSettable() Specify whether the massTarget property of the basic component is settable.

<double precision> massTargetRateGet() Returns a zero rate for the massTarget property for the basic component class.

<void> massTargetSet(<double precision> value) Set the massTarget property of the basic component.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a null implementation of the basic component class.

<logical> nonEvolvingIsActive() Return true if the nonEvolving implementation of the basic component is the active choice.

<logical> nullIsActive() Return true if the null implementation of the basic component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a basic component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a null implementation of the basic component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a null implementation of the basic component.

<void> postOutput(<double precision> time→) Perform post-output processing of a basic component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<double precision> radiusTurnaround() Returns the default value for the radiusTurnaround property for the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> radiusTurnaroundAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the radiusTurnaround property

<integer> radiusTurnaroundCount() Compute the count of evolvable quantities in the radiusTurnaround property of the BasicStandardExtended component.

<double precision> radiusTurnaroundGrowthRate() Returns the default value for the radiusTurnaroundGrowthRate property for the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> radiusTurnaroundGrowthRateAttributeMatch([requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the radiusTurnaroundGrowthRate property

<logical> radiusTurnaroundGrowthRateIsGettable() Returns true if the radiusTurnaroundGrowthRate property is gettable for the basic component class.

<logical> radiusTurnaroundGrowthRateIsSettable() Specify whether the radiusTurnaroundGrowthRate property of the basic component is settable.

<double precision> radiusTurnaroundGrowthRateRateGet() Returns a zero rate for the radiusTurnaroundGrowthRate property for the basic component class.

<void> radiusTurnaroundGrowthRateSet(<double precision> value) Set the radiusTurnaroundGrowthRate property of the basic component.

<logical> radiusTurnaroundIsGettable() Returns true if the radiusTurnaround property is gettable for the basic component class.

<logical> radiusTurnaroundIsSettable() Specify whether the radiusTurnaround property of the basic component is settable.

<void> radiusTurnaroundRate(<double precision> value) Cumulate to the rate of the radiusTurnaround property of the BasicStandardExtended component.

<double precision> radiusTurnaroundRateGet() Returns a zero rate for the radiusTurnaround property for the basic component class.

<void> radiusTurnaroundScale(<double precision> value) Set the scale of the radiusTurnaround property of the BasicStandardExtended component.

<void> radiusTurnaroundSet(<double precision> value) Set the radiusTurnaround property of the basic component.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a null implementation of the basic component.

<void> serializeASCII() Serialize the contents of a null implementation of the basic component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a null implementation of the basic component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a null implementation of the basic component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a null implementation of the basic component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a null implementation of the basic component to XML.

<integer(c_size_t)> sizeOf() Return the size in bytes of a null implementation of the basic component.

<logical> standardExtendedIsActive() Return true if the standardExtended implementation of the basic component is the active choice.

<logical> standardIsActive() Return true if the standard implementation of the basic component is the active choice.

<logical> standardTrackingIsActive() Return true if the standardTracking implementation of the basic component is the active choice.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<double precision> time() Returns the default value for the time property for the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> timeAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the time property

<integer> timeCount() Compute the count of evolvable quantities in the time property of the BasicNonEvolving component.

<logical> timeIsGettable() Returns true if the time property is gettable for the basic component class.

<logical> timeIsSettable() Specify whether the time property of the basic component is settable.

<double precision> timeLastIsolated() Returns the default value for the timeLastIsolated property for the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> timeLastIsolatedAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the timeLastIsolated property

<logical> timeLastIsolatedIsGettable() Returns true if the timeLastIsolated property is gettable for the basic component class.

<logical> timeLastIsolatedIsSettable() Specify whether the timeLastIsolated property of the basic component is settable.

<double precision> timeLastIsolatedRateGet() Returns a zero rate for the timeLastIsolated property for the basic component class.

<void> timeLastIsolatedSet(<double precision> value) Set the timeLastIsolated property of the basic component.

<void> timeRate(<double precision> value) Cumulate to the rate of the time property of the BasicNonEvolving component.

<double precision> timeRateGet() Returns a zero rate for the time property for the basic component class.

<void> timeScale(<double precision> value) Set the scale of the time property of the BasicNonEvolving component.

<void> timeSet(<double precision> value) Set the time property of the basic component.

<type(varying_string)> type() Returns the type name for the null implementation of the basic component class.

nodeComponentBasicStandard

<double precision> accretionRate() Get the accretionRate property of an standard implementation of the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> accretionRateAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the accretionRate property

<double precision> accretionRateBertschinger() Returns the default value for the accretionRateBertschinger property for the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> accretionRateBertschingerAttributeMatch(< [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the accretionRateBertschinger property

<logical> accretionRateBertschingerIsGettable() Returns true if the accretionRateBertschinger property is gettable for the basic component class.

<logical> accretionRateBertschingerIsSettable() Specify whether the accretionRateBertschinger property of the basic component is settable.

<double precision> accretionRateBertschingerRateGet() Returns a zero rate for the accretionRateBertschinger property for the basic component class.

<void> accretionRateBertschingerSet(<double precision> value) Set the accretionRateBertschinger property of the basic component.

<logical> accretionRateIsGettable() Returns true if the accretionRate property is gettable for the basic component class.

<logical> accretionRateIsSettable() Specify whether the accretionRate property of the basic component is settable.

<double precision> accretionRateRateGet() Returns a zero rate for the accretionRate property for the basic component class.

<void> accretionRateSet(<double precision> setValue→) Set the accretionRate property of an standard implementation of the basic component class.

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a standard implementation of the basic component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a standard implementation of the basic component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a standard implementation of the basic component from array.

<void> destroy() Finalize a standard implementation of the basic component.

<void> dumpASCII() Dump the content of a basic component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<logical> extendedTrackingIsActive() Return true if the extendedTracking implementation of the basic component is the active choice.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a standard member of the basic component.

<double precision> mass() Get the mass property of an standard implementation of the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> massAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the mass property

<double precision> massBertschinger() Returns the default value for the massBertschinger property for the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> massBertschingerAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the massBertschinger property

<integer> massBertschingerCount() Compute the count of evolvable quantities in the massBertschinger property of the BasicStandardExtended component.

<logical> massBertschingerIsGettable() Returns true if the massBertschinger property is gettable for the basic component class.

<logical> massBertschingerIsSettable() Specify whether the massBertschinger property of the basic component is settable.

<void> massBertschingerRate(<double precision> value) Cumulate to the rate of the massBertschinger property of the BasicStandardExtended component.

<double precision> massBertschingerRateGet() Returns a zero rate for the massBertschinger property for the basic component class.

<void> massBertschingerScale(<double precision> value) Set the scale of the massBertschinger property of the BasicStandardExtended component.

<void> massBertschingerSet(<double precision> value) Set the massBertschinger property of the basic component.

<integer> massCount() Return a count of the number of scalar properties in the mass property of an standard implementation of the basic component class.

<void> massInactive() Indicate that the mass property of an standard implementation of the basic component class is inactive for differential equation solving.

<logical> massIsGettable() Returns true if the mass property is gettable for the basic component class.

<logical> massIsSettable() Specify whether the mass property of the basic component is settable.

<double precision> massMaximum() Returns the default value for the massMaximum property for the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> massMaximumAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the massMaximum property

<logical> massMaximumIsGettable() Returns true if the massMaximum property is gettable for the basic component class.

<logical> massMaximumIsSettable() Specify whether the massMaximum property of the basic component is settable.

<double precision> massMaximumRateGet() Returns a zero rate for the massMaximum property for the basic component class.

<void> massMaximumSet(<double precision> value) Set the massMaximum property of the basic component.

<void> massRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the mass property of an standard implementation of the basic component class.

<double precision> massRateGet() Get the rate of change of the mass property of an standard implementation of the basic component class.

<void> massScale(<double precision> setValue→) Set the absolute scale of the mass property of an standard implementation of the basic component class.

<void> massSet(<double precision> setValue→) Set the mass property of an standard implementation of the basic component class.

<double precision> massTarget() Get the massTarget property of an standard implementation of the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> massTargetAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the massTarget property

<logical> massTargetIsGettable() Returns true if the massTarget property is gettable for the basic component class.

<logical> massTargetIsSettable() Specify whether the massTarget property of the basic component is settable.

<double precision> massTargetRateGet() Returns a zero rate for the massTarget property for the basic component class.

<void> massTargetSet(<double precision> setValue→) Set the massTarget property of an standard implementation of the basic component class.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a standard implementation of the basic component class.

<logical> nonEvolvingIsActive() Return true if the nonEvolving implementation of the basic component is the active choice.

<logical> nullIsActive() Return true if the null implementation of the basic component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a basic component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a standard implementation of the basic component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a standard implementation of the basic component.

<void> postOutput(<double precision> time→) Perform post-output processing of a basic component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<double precision> radiusTurnaround() Returns the default value for the radiusTurnaround property for the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> radiusTurnaroundAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the radiusTurnaround property

<integer> radiusTurnaroundCount() Compute the count of evolvable quantities in the radiusTurnaround property of the BasicStandardExtended component.

<double precision> radiusTurnaroundGrowthRate() Returns the default value for the radiusTurnaroundGrowthRate property for the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> radiusTurnaroundGrowthRateAttributeMatch([requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the radiusTurnaroundGrowthRate property

<logical> radiusTurnaroundGrowthRateIsGettable() Returns true if the radiusTurnaroundGrowthRate property is gettable for the basic component class.

<logical> radiusTurnaroundGrowthRateIsSettable() Specify whether the radiusTurnaroundGrowthRate property of the basic component is settable.

<double precision> radiusTurnaroundGrowthRateRateGet() Returns a zero rate for the radiusTurnaroundGrowthRate property for the basic component class.

<void> radiusTurnaroundGrowthRateSet(<double precision> value) Set the radiusTurnaroundGrowthRate property of the basic component.

<logical> radiusTurnaroundIsGettable() Returns true if the radiusTurnaround property is gettable for the basic component class.

<logical> radiusTurnaroundIsSettable() Specify whether the radiusTurnaround property of the basic component is settable.

<void> radiusTurnaroundRate(<double precision> value) Cumulate to the rate of the radiusTurnaround property of the BasicStandardExtended component.

<double precision> radiusTurnaroundRateGet() Returns a zero rate for the radiusTurnaround property for the basic component class.

<void> radiusTurnaroundScale(<double precision> value) Set the scale of the radiusTurnaround property of the BasicStandardExtended component.

<void> radiusTurnaroundSet(<double precision> value) Set the radiusTurnaround property of the basic component.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a standard implementation of the basic component.

<void> serializeASCII() Serialize the contents of a standard implementation of the basic component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a standard implementation of the basic component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a standard implementation of the basic component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a standard implementation of the basic component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a standard implementation of the basic component to XML.

<integer(c_size_t)> sizeOf() Return the size in bytes of a standard implementation of the basic component.

<logical> standardExtendedIsActive() Return true if the standardExtended implementation of the basic component is the active choice.

<logical> standardIsActive() Return true if the standard implementation of the basic component is the active choice.

<logical> standardTrackingIsActive() Return true if the standardTracking implementation of the basic component is the active choice.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<double precision> time() Get the time property of an standard implementation of the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> timeAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the time property

<integer> timeCount() Return a count of the number of scalar properties in the time property of an standard implementation of the basic component class.

<void> timeInactive() Indicate that the time property of an standard implementation of the basic component class is inactive for differential equation solving.

<logical> timeIsGettable() Returns true if the time property is gettable for the basic component class.

<logical> timeIsSettable() Specify whether the time property of the basic component is settable.

<double precision> timeLastIsolated() Returns the default value for the timeLastIsolated property for the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> timeLastIsolatedAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the timeLastIsolated property

<logical> timeLastIsolatedIsGettable() Returns true if the timeLastIsolated property is gettable for the basic component class.

<logical> timeLastIsolatedIsSettable() Specify whether the timeLastIsolated property of the basic component is settable.

<double precision> timeLastIsolatedRateGet() Returns a zero rate for the timeLastIsolated property for the basic component class.

<void> timeLastIsolatedSet(<double precision> setValue→) Set the timeLastIsolated property of an standard implementation of the basic component class.

<void> timeRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the time property of an standard implementation of the basic component class.

<double precision> timeRateGet() Get the rate of change of the time property of an standard implementation of the basic component class.

<void> timeScale(<double precision> setValue→) Set the absolute scale of the time property of an standard implementation of the basic component class.

<void> timeSet(<double precision> setValue→) Set the time property of an standard implementation of the basic component class.

<type(varying_string)> type() Returns the type name for the standard implementation of the basic component class.

nodeComponentBasicStandardExtended

<double precision> accretionRate() Get the accretionRate property of an standard implementation of the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> accretionRateAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the accretionRate property

<double precision> accretionRateBertschinger() Get the accretionRateBertschinger property of an standardExtended implementation of the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> accretionRateBertschingerAttributeMatch(< [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the accretionRateBertschinger property

<logical> accretionRateBertschingerIsGettable() Returns true if the accretionRateBertschinger property is gettable for the basic component class.

<logical> accretionRateBertschingerIsSettable() Specify whether the accretionRateBertschinger property of the basic component is settable.

<double precision> accretionRateBertschingerRateGet() Returns a zero rate for the accretionRateBertschinger property for the basic component class.

<void> accretionRateBertschingerSet(<double precision> setValue→) Set the accretionRateBertschinger property of an standardExtended implementation of the basic component class.

<logical> accretionRateIsGettable() Returns true if the accretionRate property is gettable for the basic component class.

<logical> accretionRateIsSettable() Specify whether the accretionRate property of the basic component is settable.

<double precision> accretionRateRateGet() Returns a zero rate for the accretionRate property for the basic component class.

<void> accretionRateSet(<double precision> setValue→) Set the accretionRate property of an standard implementation of the basic component class.

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a standardExtended implementation of the basic component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a standardExtended implementation of the basic component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a standardExtended implementation of the basic component from array.

<void> destroy() Finalize a standardExtended implementation of the basic component.

<void> dumpASCII() Dump the content of a basic component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<logical> extendedTrackingIsActive() Return true if the extendedTracking implementation of the basic component is the active choice.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a standardExtended member of the basic component.

<double precision> mass() Get the mass property of an standard implementation of the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> massAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the mass property

<double precision> massBertschinger() Get the value of the massBertschinger property of the standardExtended implementation of the basic component using a deferred function.

<type(varying_string), allocatable, dimension(:) => matches> massBertschingerAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the massBertschinger property

<integer> massBertschingerCount() Return a count of the number of scalar properties in the massBertschinger property of an standardExtended implementation of the basic component class.

<void> massBertschingerFunction() Set the function to be used for the get method of the massBertschinger property of the BasicStandardExtended component.

<void> massBertschingerInactive() Indicate that the massBertschinger property of an standardExtended implementation of the basic component class is inactive for differential equation solving.

<logical> massBertschingerIsAttached() Return true if the deferred function used to get the massBertschinger property of the BasicStandardExtended component class has been attached.

<logical> massBertschingerIsGettable() Returns true if the massBertschinger property is gettable for the basic component class.

<logical> massBertschingerIsSettable() Specify whether the massBertschinger property of the basic component is settable.

<void> massBertschingerRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interru [interruptProcedure]↔) Accumulate to the rate of change of the massBertschinger property of an standardExtended implementation of the basic component class.

<double precision> massBertschingerRateGet() Get the rate of change of the massBertschinger property of an standardExtended implementation of the basic component class.

<void> massBertschingerScale(<double precision> setValue→) Set the absolute scale of the massBertschinger property of an standardExtended implementation of the basic component class.

<void> massBertschingerSet(<double precision> setValue→) Set the massBertschinger property of an standardExtended implementation of the basic component class.

<double precision> massBertschingerValue() Get the massBertschinger property of an standardExtended implementation of the basic component class.

<integer> massCount() Return a count of the number of scalar properties in the mass property of an standard implementation of the basic component class.

<void> massInactive() Indicate that the mass property of an standard implementation of the basic component class is inactive for differential equation solving.

<logical> massIsGettable() Returns true if the mass property is gettable for the basic component class.

<logical> massIsSettable() Specify whether the mass property of the basic component is settable.

<double precision> massMaximum() Returns the default value for the massMaximum property for the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> massMaximumAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the massMaximum property

<logical> massMaximumIsGettable() Returns true if the massMaximum property is gettable for the basic component class.

<logical> massMaximumIsSettable() Specify whether the massMaximum property of the basic component is settable.

<double precision> massMaximumRateGet() Returns a zero rate for the massMaximum property for the basic component class.

<void> massMaximumSet(<double precision> value) Set the massMaximum property of the basic component.

<void> massRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the mass property of an standard implementation of the basic component class.

<double precision> massRateGet() Get the rate of change of the mass property of an standard implementation of the basic component class.

<void> massScale(<double precision> setValue→) Set the absolute scale of the mass property of an standard implementation of the basic component class.

<void> massSet(<double precision> setValue→) Set the mass property of an standard implementation of the basic component class.

<double precision> massTarget() Get the massTarget property of an standard implementation of the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> massTargetAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the massTarget property

<logical> massTargetIsGettable() Returns true if the massTarget property is gettable for the basic component class.

<logical> massTargetIsSettable() Specify whether the massTarget property of the basic component is settable.

<double precision> massTargetRateGet() Returns a zero rate for the massTarget property for the basic component class.

<void> massTargetSet(<double precision> setValue→) Set the massTarget property of an standard implementation of the basic component class.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a standardExtended implementation of the basic component class.

<logical> nonEvolvingIsActive() Return true if the nonEvolving implementation of the basic component is the active choice.

<logical> nullIsActive() Return true if the null implementation of the basic component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a basic component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a standardExtended implementation of the basic component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a standardExtended implementation of the basic component.

<void> postOutput(<double precision> time→) Perform post-output processing for a standardExtended implementation of the basic component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<double precision> radiusTurnaround() Get the value of the radiusTurnaround property of the standardExtended implementation of the basic component using a deferred function.

<type(varying_string), allocatable, dimension(:) => matches> radiusTurnaroundAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the radiusTurnaround property

<integer> radiusTurnaroundCount() Return a count of the number of scalar properties in the radiusTurnaround property of an standardExtended implementation of the basic component class.

<void> radiusTurnaroundFunction() Set the function to be used for the get method of the radiusTurnaround property of the BasicStandardExtended component.

<double precision> radiusTurnaroundGrowthRate() Get the radiusTurnaroundGrowthRate property of an standardExtended implementation of the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> radiusTurnaroundGrowthRateAttributeMatch([requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the radiusTurnaroundGrowthRate property

<logical> radiusTurnaroundGrowthRateIsGettable() Returns true if the radiusTurnaroundGrowthRate property is gettable for the basic component class.

<logical> radiusTurnaroundGrowthRateIsSettable() Specify whether the radiusTurnaroundGrowthRate property of the basic component is settable.

<double precision> radiusTurnaroundGrowthRateRateGet() Returns a zero rate for the radiusTurnaroundGrowthRate property for the basic component class.

<void> radiusTurnaroundGrowthRateSet(<double precision> setValue→) Set the radiusTurnaroundGrowthRate property of an standardExtended implementation of the basic component class.

<void> radiusTurnaroundInactive() Indicate that the radiusTurnaround property of an standardExtended implementation of the basic component class is inactive for differential equation solving.

<logical> radiusTurnaroundIsAttached() Return true if the deferred function used to get the radiusTurnaround property of the BasicStandardExtended component class has been attached.

<logical> radiusTurnaroundIsGettable() Returns true if the radiusTurnaround property is gettable for the basic component class.

<logical> radiusTurnaroundIsSettable() Specify whether the radiusTurnaround property of the basic component is settable.

<void> radiusTurnaroundRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interru [interruptProcedure]↔) Accumulate to the rate of change of the radiusTurnaround property of an standardExtended implementation of the basic component class.

<double precision> radiusTurnaroundRateGet() Get the rate of change of the radiusTurnaround property of an standardExtended implementation of the basic component class.

<void> radiusTurnaroundScale(<double precision> setValue→) Set the absolute scale of the radiusTurnaround property of an standardExtended implementation of the basic component class.

<void> radiusTurnaroundSet(<double precision> setValue→) Set the radiusTurnaround property of an standardExtended implementation of the basic component class.

<double precision> radiusTurnaroundValue() Get the radiusTurnaround property of an standardExtended implementation of the basic component class.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a standardExtended implementation of the basic component.

<void> serializeASCII() Serialize the contents of a standardExtended implementation of the basic component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a standardExtended implementation of the basic component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a standardExtended implementation of the basic component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a standardExtended implementation of the basic component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a standardExtended implementation of the basic component to XML.

<integer(c_size_t)> sizeOf() Return the size in bytes of a standardExtended implementation of the basic component.

<logical> standardExtendedIsActive() Return true if the standardExtended implementation of the basic component is the active choice.

<logical> standardIsActive() Return true if the standard implementation of the basic component is the active choice.

<logical> standardTrackingIsActive() Return true if the standardTracking implementation of the basic component is the active choice.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<double precision> time() Get the time property of an standard implementation of the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> timeAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the time property

<integer> timeCount() Return a count of the number of scalar properties in the time property of an standard implementation of the basic component class.

<void> timeInactive() Indicate that the time property of an standard implementation of the basic component class is inactive for differential equation solving.

<logical> timeIsGettable() Returns true if the time property is gettable for the basic component class.

<logical> timeIsSettable() Specify whether the time property of the basic component is settable.

<double precision> timeLastIsolated() Returns the default value for the timeLastIsolated property for the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> timeLastIsolatedAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the timeLastIsolated property

<logical> timeLastIsolatedIsGettable() Returns true if the timeLastIsolated property is gettable for the basic component class.

<logical> timeLastIsolatedIsSettable() Specify whether the timeLastIsolated property of the basic component is settable.

<double precision> timeLastIsolatedRateGet() Returns a zero rate for the timeLastIsolated property for the basic component class.

<void> timeLastIsolatedSet(<double precision> setValue→) Set the timeLastIsolated property of an standard implementation of the basic component class.

<void> timeRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the time property of an standard implementation of the basic component class.

<double precision> timeRateGet() Get the rate of change of the time property of an standard implementation of the basic component class.

<void> timeScale(<double precision> setValue→) Set the absolute scale of the time property of an standard implementation of the basic component class.

<void> timeSet(<double precision> setValue→) Set the time property of an standard implementation of the basic component class.

<type(varying_string)> type() Returns the type name for the standardExtended implementation of the basic component class.

nodeComponentBasicStandardTracking

<double precision> accretionRate() Get the accretionRate property of an standard implementation of the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> accretionRateAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the accretionRate property

<double precision> accretionRateBertschinger() Returns the default value for the accretionRateBertschinger property for the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> accretionRateBertschingerAttributeMatch(< [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the accretionRateBertschinger property

<logical> accretionRateBertschingerIsGettable() Returns true if the accretionRateBertschinger property is gettable for the basic component class.

<logical> accretionRateBertschingerIsSettable() Specify whether the accretionRateBertschinger property of the basic component is settable.

<double precision> accretionRateBertschingerRateGet() Returns a zero rate for the accretionRateBertschinger property for the basic component class.

<void> accretionRateBertschingerSet(<double precision> value) Set the accretionRateBertschinger property of the basic component.

<logical> accretionRateIsGettable() Returns true if the accretionRate property is gettable for the basic component class.

<logical> accretionRateIsSettable() Specify whether the accretionRate property of the basic component is settable.

<double precision> accretionRateRateGet() Returns a zero rate for the accretionRate property for the basic component class.

<void> accretionRateSet(<double precision> setValue→) Set the accretionRate property of an standard implementation of the basic component class.

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a standardTracking implementation of the basic component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a standardTracking implementation of the basic component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a standardTracking implementation of the basic component from array.

<void> destroy() Finalize a standardTracking implementation of the basic component.

<void> dumpASCII() Dump the content of a basic component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<logical> extendedTrackingIsActive() Return true if the extendedTracking implementation of the basic component is the active choice.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a standardTracking member of the basic component.

<double precision> mass() Get the mass property of an standard implementation of the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> massAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the mass property

<double precision> massBertschinger() Returns the default value for the massBertschinger property for the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> massBertschingerAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the massBertschinger property

<integer> massBertschingerCount() Compute the count of evolvable quantities in the massBertschinger property of the BasicStandardExtended component.

<logical> massBertschingerIsGettable() Returns true if the massBertschinger property is gettable for the basic component class.

<logical> massBertschingerIsSettable() Specify whether the massBertschinger property of the basic component is settable.

<void> massBertschingerRate(<double precision> value) Cumulate to the rate of the massBertschinger property of the BasicStandardExtended component.

<double precision> massBertschingerRateGet() Returns a zero rate for the massBertschinger property for the basic component class.

<void> massBertschingerScale(<double precision> value) Set the scale of the massBertschinger property of the BasicStandardExtended component.

<void> massBertschingerSet(<double precision> value) Set the massBertschinger property of the basic component.

<integer> massCount() Return a count of the number of scalar properties in the mass property of an standard implementation of the basic component class.

<void> massInactive() Indicate that the mass property of an standard implementation of the basic component class is inactive for differential equation solving.

<logical> massIsGettable() Returns true if the mass property is gettable for the basic component class.

<logical> massIsSettable() Specify whether the mass property of the basic component is settable.

<double precision> massMaximum() Get the massMaximum property of an standardTracking implementation of the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> massMaximumAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the massMaximum property

<logical> massMaximumIsGettable() Returns true if the massMaximum property is gettable for the basic component class.

<logical> massMaximumIsSettable() Specify whether the massMaximum property of the basic component is settable.

<double precision> massMaximumRateGet() Returns a zero rate for the massMaximum property for the basic component class.

<void> massMaximumSet(<double precision> setValue→) Set the massMaximum property of an standardTracking implementation of the basic component class.

<void> massRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the mass property of an standard implementation of the basic component class.

<double precision> massRateGet() Get the rate of change of the mass property of an standard implementation of the basic component class.

<void> massScale(<double precision> setValue→) Set the absolute scale of the mass property of an standard implementation of the basic component class.

<void> massSet(<double precision> setValue→) Set the mass property of an standard implementation of the basic component class.

<double precision> massTarget() Get the massTarget property of an standard implementation of the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> massTargetAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the massTarget property

<logical> massTargetIsGettable() Returns true if the massTarget property is gettable for the basic component class.

<logical> massTargetIsSettable() Specify whether the massTarget property of the basic component is settable.

<double precision> massTargetRateGet() Returns a zero rate for the massTarget property for the basic component class.

<void> massTargetSet(<double precision> setValue→) Set the massTarget property of an standard implementation of the basic component class.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a standardTracking implementation of the basic component class.

<logical> nonEvolvingIsActive() Return true if the nonEvolving implementation of the basic component is the active choice.

<logical> nullIsActive() Return true if the null implementation of the basic component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a basic component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a standardTracking implementation of the basic component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a standardTracking implementation of the basic component.

<void> postOutput(<double precision> time→) Perform post-output processing for a standardTracking implementation of the basic component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<double precision> radiusTurnaround() Returns the default value for the radiusTurnaround property for the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> radiusTurnaroundAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the radiusTurnaround property

<integer> radiusTurnaroundCount() Compute the count of evolvable quantities in the radiusTurnaround property of the BasicStandardExtended component.

<double precision> radiusTurnaroundGrowthRate() Returns the default value for the radiusTurnaroundGrowthRate property for the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> radiusTurnaroundGrowthRateAttributeMatch([requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the radiusTurnaroundGrowthRate property

<logical> radiusTurnaroundGrowthRateIsGettable() Returns true if the radiusTurnaroundGrowthRate property is gettable for the basic component class.

<logical> radiusTurnaroundGrowthRateIsSettable() Specify whether the radiusTurnaroundGrowthRate property of the basic component is settable.

<double precision> radiusTurnaroundGrowthRateRateGet() Returns a zero rate for the radiusTurnaroundGrowthRate property for the basic component class.

<void> radiusTurnaroundGrowthRateSet(<double precision> value) Set the radiusTurnaroundGrowthRate property of the basic component.

<logical> radiusTurnaroundIsGettable() Returns true if the radiusTurnaround property is gettable for the basic component class.

<logical> radiusTurnaroundIsSettable() Specify whether the radiusTurnaround property of the basic component is settable.

<void> radiusTurnaroundRate(<double precision> value) Cumulate to the rate of the radiusTurnaround property of the BasicStandardExtended component.

<double precision> radiusTurnaroundRateGet() Returns a zero rate for the radiusTurnaround property for the basic component class.

<void> radiusTurnaroundScale(<double precision> value) Set the scale of the radiusTurnaround property of the BasicStandardExtended component.

<void> radiusTurnaroundSet(<double precision> value) Set the radiusTurnaround property of the basic component.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a standardTracking implementation of the basic component.

<void> serializeASCII() Serialize the contents of a standardTracking implementation of the basic component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a standardTracking implementation of the basic component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a standardTracking implementation of the basic component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a standardTracking implementation of the basic component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a standardTracking implementation of the basic component to XML.

<integer(c_size_t)> sizeOf() Return the size in bytes of a standardTracking implementation of the basic component.

<logical> standardExtendedIsActive() Return true if the standardExtended implementation of the basic component is the active choice.

<logical> standardIsActive() Return true if the standard implementation of the basic component is the active choice.

<logical> standardTrackingIsActive() Return true if the standardTracking implementation of the basic component is the active choice.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<double precision> time() Get the time property of an standard implementation of the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> timeAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the time property

<integer> timeCount() Return a count of the number of scalar properties in the time property of an standard implementation of the basic component class.

<void> timeInactive() Indicate that the time property of an standard implementation of the basic component class is inactive for differential equation solving.

<logical> timeIsGettable() Returns true if the time property is gettable for the basic component class.

<logical> timeIsSettable() Specify whether the time property of the basic component is settable.

<double precision> timeLastIsolated() Returns the default value for the timeLastIsolated property for the basic component class.

<type(varying_string), allocatable, dimension(:) => matches> timeLastIsolatedAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the basic class that have the desired attributes for the timeLastIsolated property

<logical> timeLastIsolatedIsGettable() Returns true if the timeLastIsolated property is gettable for the basic component class.

<logical> timeLastIsolatedIsSettable() Specify whether the timeLastIsolated property of the basic component is settable.

<double precision> timeLastIsolatedRateGet() Returns a zero rate for the timeLastIsolated property for the basic component class.

<void> timeLastIsolatedSet(<double precision> setValue→) Set the timeLastIsolated property of an standard implementation of the basic component class.

<void> timeRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the time property of an standard implementation of the basic component class.

<double precision> timeRateGet() Get the rate of change of the time property of an standard implementation of the basic component class.

<void> timeScale(<double precision> setValue→) Set the absolute scale of the time property of an standard implementation of the basic component class.

<void> timeSet(<double precision> setValue→) Set the time property of an standard implementation of the basic component class.

<type(varying_string)> type() Returns the type name for the standardTracking implementation of the basic component class.

nodeComponentBlackHole

<double precision> accretionRate() Returns the default value for the accretionRate property for the blackHole component class.

<type(varying_string), allocatable, dimension(:) => matches> accretionRateAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the blackHole class that have the desired attributes for the accretionRate property

<logical> accretionRateIsGettable() Returns true if the accretionRate property is gettable for the blackHole component class.

<logical> accretionRateIsSettable() Specify whether the accretionRate property of the blackHole component is settable.

<double precision> accretionRateRateGet() Returns a zero rate for the accretionRate property for the blackHole component class.

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a generic blackHole component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Read properties from raw file.

<void> deserializeValues(<double(:)> array→, <integer> propertyType→) Deserialize the evolvable quantities from an array.

<void> destroy() Finalize a generic blackHole component.

<void> dumpASCII() Dump the content of a blackHole component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a generic blackHole component.

<double precision> mass() Returns the default value for the mass property for the blackHole component class.

<type(varying_string), allocatable, dimension(:) => matches> massAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the blackHole class that have the desired attributes for the mass property

<integer> massCount() Compute the count of evolvable quantities in the mass property of the BlackHoleSimple component.

<logical> massIsGettable() Returns true if the mass property is gettable for the blackHole component class.

<logical> massIsSettable() Specify whether the mass property of the blackHole component is settable.

<void> massRate(<double precision> value) Cumulate to the rate of the mass property of the BlackHoleSimple component.

<double precision> massRateGet() Returns a zero rate for the mass property for the blackHole component class.

<void> massScale(<double precision> value) Set the scale of the mass property of the BlackHoleSimple component.

<double precision> massSeed() Returns the default value for the massSeed property for the blackHole component class.

<type(varying_string), allocatable, dimension(:) => matches> massSeedAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the blackHole class that have the desired attributes for the massSeed property

<logical> massSeedIsGettable() Returns true if the massSeed property is gettable for the blackHole component class.

<logical> massSeedIsSettable() Specify whether the massSeed property of the blackHole component is settable.

<double precision> massSeedRateGet() Returns a zero rate for the massSeed property for the blackHole component class.

<void> massSet(<double precision> value) Set the mass property of the blackHole component.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a nodeComponent object.

<logical> nonCentralIsActive() Return true if the nonCentral implementation of the blackHole component is the active choice.

<logical> nullIsActive() Return true if the null implementation of the blackHole component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a blackHole component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a generic blackHole component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Establish the names of properties to output for a generic blackHole component.

<void> postOutput(<double precision> time→) Perform post-output processing of a blackHole component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<double precision> radialPosition() Returns the default value for the radialPosition property for the blackHole component class.

<type(varying_string), allocatable, dimension(:) => matches> radialPositionAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the blackHole class that have the desired attributes for the radialPosition property

<integer> radialPositionCount() Compute the count of evolvable quantities in the radialPosition property of the BlackHoleNonCentral component.

<logical> radialPositionIsGettable() Returns true if the radialPosition property is gettable for the blackHole component class.

<logical> radialPositionIsSettable() Specify whether the radialPosition property of the blackHole component is settable.

<void> radialPositionRate(<double precision> value) Cumulate to the rate of the radialPosition property of the BlackHoleNonCentral component.

<double precision> radialPositionRateGet() Returns a zero rate for the radialPosition property for the blackHole component class.

<void> radialPositionScale(<double precision> value) Set the scale of the radialPosition property of the BlackHoleNonCentral component.

<void> radialPositionSet(<double precision> value) Set the radialPosition property of the blackHole component.

<double precision> radiativeEfficiency() Returns the default value for the radiativeEfficiency property for the blackHole component class.

<type(varying_string), allocatable, dimension(:) => matches> radiativeEfficiencyAttributeMatch(<logica [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the blackHole class that have the desired attributes for the radiativeEfficiency property

<logical> radiativeEfficiencyIsGettable() Returns true if the radiativeEfficiency property is gettable for the blackHole component class.

<logical> radiativeEfficiencyIsSettable() Specify whether the radiativeEfficiency property of the blackHole component is settable.

<double precision> radiativeEfficiencyRateGet() Returns a zero rate for the radiativeEfficiency property for the blackHole component class.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets() Set offsets into serialization arrays.

<void> serializeASCII() Serialize the content of a blackHole component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the number of evolvable quantities to be evolved.

<void> serializeRaw(<integer> fileHandle→) Generate a binary dump of all properties.

<void> serializeValues(<double(:)> array←, <integer> propertyType→) Serialize the evolvable quantities to an array.

<void> serializeXML() Generate an XML dump of all properties.

<logical> simpleIsActive() Return true if the simple implementation of the blackHole component is the active choice.

<integer(c_size_t)> sizeOf() Return the size in bytes of a nodeComponentBlackHole component.

<double precision> spin() Returns the default value for the spin property for the blackHole component class.

<type(varying_string), allocatable, dimension(:) => matches> spinAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the blackHole class that have the desired attributes for the spin property

<integer> spinCount() Compute the count of evolvable quantities in the spin property of the BlackHoleStandard component.

<logical> spinIsGettable() Returns true if the spin property is gettable for the blackHole component class.

<logical> spinIsSettable() Specify whether the spin property of the blackHole component is settable.

<void> spinRate(<double precision> value) Cumulate to the rate of the spin property of the BlackHoleStandard component.

<double precision> spinRateGet() Returns a zero rate for the spin property for the blackHole component class.

<void> spinScale(<double precision> value) Set the scale of the spin property of the BlackHoleStandard component.

<double precision> spinSeed() Returns the default value for the spinSeed property for the blackHole component class.

<type(varying_string), allocatable, dimension(:) => matches> spinSeedAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the blackHole class that have the desired attributes for the spinSeed property

<logical> spinSeedIsGettable() Returns true if the spinSeed property is gettable for the blackHole component class.

<logical> spinSeedIsSettable() Specify whether the spinSeed property of the blackHole component is settable.

<double precision> spinSeedRateGet() Returns a zero rate for the spinSeed property for the blackHole component class.

<void> spinSet(<double precision> value) Set the spin property of the blackHole component.

<logical> standardIsActive() Return true if the standard implementation of the blackHole component is the active choice.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<double precision> tripleInteractionTime() Returns the default value for the tripleInteractionTime property for the blackHole component class.

<type(varying_string), allocatable, dimension(:) => matches> tripleInteractionTimeAttributeMatch(<logi
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the blackHole class that have the desired attributes for the tripleInteractionTime property

<logical> tripleInteractionTimeIsGettable() Returns true if the tripleInteractionTime property is gettable for the blackHole component class.

<logical> tripleInteractionTimeIsSettable() Specify whether the tripleInteractionTime property of the blackHole component is settable.

<double precision> tripleInteractionTimeRateGet() Returns a zero rate for the tripleInteractionTime property for the blackHole component class.

<void> tripleInteractionTimeSet(<double precision> value) Set the tripleInteractionTime property of the blackHole component.

<type(varying_string)> type() Returns the type name for the blackHole component class.

nodeComponentBlackHoleNonCentral

<double precision> accretionRate() Get the value of the accretionRate property of the standard implementation of the blackHole component using a deferred function.

<type(varying_string), allocatable, dimension(:) => matches> accretionRateAttributeMatch(<logical>
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the blackHole class that have the desired attributes for the accretionRate property

<void> accretionRateFunction() Set the function to be used for the get method of the accretionRate property of the BlackHoleStandard component.

<logical> accretionRateIsAttached() Return true if the deferred function used to get the accretionRate property of the BlackHoleStandard component class has been attached.

<logical> accretionRateIsGettable() Returns true if the accretionRate property is gettable for the blackHole component class.

<logical> accretionRateIsSettable() Specify whether the accretionRate property of the blackHole component is settable.

<double precision> accretionRateRateGet() Returns a zero rate for the accretionRate property for the blackHole component class.

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a nonCentral implementation of the blackHole component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType>
[massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a nonCentral implementation of the blackHole component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a nonCentral implementation of the blackHole component from array.

<void> destroy() Finalize a nonCentral implementation of the blackHole component.

<void> dumpASCII() Dump the content of a blackHole component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter
[interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a nonCentral member of the blackHole component.

<double precision> mass() Get the mass property of an standard implementation of the blackHole component class.

<type(varying_string), allocatable, dimension(:) => matches> massAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the blackHole class that have the desired attributes for the mass property

<integer> massCount() Return a count of the number of scalar properties in the mass property of an standard implementation of the blackHole component class.

<void> massInactive() Indicate that the mass property of an standard implementation of the blackHole component class is inactive for differential equation solving.

<logical> massIsGettable() Returns true if the mass property is gettable for the blackHole component class.

<logical> massIsSettable() Specify whether the mass property of the blackHole component is settable.

<void> massRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the mass property of an standard implementation of the blackHole component class.

<double precision> massRateGet() Get the rate of change of the mass property of an standard implementation of the blackHole component class.

<void> massScale(<double precision> setValue→) Set the absolute scale of the mass property of an standard implementation of the blackHole component class.

<double precision> massSeed() Returns the default value for the massSeed property for the blackHole component class.

<type(varying_string), allocatable, dimension(:) => matches> massSeedAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the blackHole class that have the desired attributes for the massSeed property

<logical> massSeedIsGettable() Returns true if the massSeed property is gettable for the blackHole component class.

<logical> massSeedIsSettable() Specify whether the massSeed property of the blackHole component is settable.

<double precision> massSeedRateGet() Returns a zero rate for the massSeed property for the blackHole component class.

<void> massSet(<double precision> setValue→) Set the mass property of an standard implementation of the blackHole component class.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a nonCentral implementation of the blackHole component class.

<logical> nonCentralIsActive() Return true if the nonCentral implementation of the blackHole component is the active choice.

<logical> nullIsActive() Return true if the null implementation of the blackHole component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a blackHole component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a nonCentral implementation of the blackHole component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a nonCentral implementation of the blackHole component.

<void> postOutput(<double precision> time→) Perform post-output processing for a nonCentral implementation of the blackHole component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<double precision> radialPosition() Get the radialPosition property of an nonCentral implementation of the blackHole component class.

<type(varying_string), allocatable, dimension(:) => matches> radialPositionAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the blackHole class that have the desired attributes for the radialPosition property

<integer> radialPositionCount() Return a count of the number of scalar properties in the radialPosition property of an nonCentral implementation of the blackHole component class.

<void> radialPositionInactive() Indicate that the radialPosition property of an nonCentral implementation of the blackHole component class is inactive for differential equation solving.

<logical> radialPositionIsGettable() Returns true if the radialPosition property is gettable for the blackHole component class.

<logical> radialPositionIsSettable() Specify whether the radialPosition property of the blackHole component is settable.

<void> radialPositionRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interrupt [interruptProcedure]↔) Accumulate to the rate of change of the radialPosition property of an nonCentral implementation of the blackHole component class.

<double precision> radialPositionRateGet() Get the rate of change of the radialPosition property of an nonCentral implementation of the blackHole component class.

<void> radialPositionScale(<double precision> setValue→) Set the absolute scale of the radialPosition property of an nonCentral implementation of the blackHole component class.

<void> radialPositionSet(<double precision> setValue→) Set the radialPosition property of an nonCentral implementation of the blackHole component class.

<double precision> radiativeEfficiency() Get the value of the radiativeEfficiency property of the standard implementation of the blackHole component using a deferred function.

<type(varying_string), allocatable, dimension(:) => matches> radiativeEfficiencyAttributeMatch(<logica [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the blackHole class that have the desired attributes for the radiativeEfficiency property

<void> radiativeEfficiencyFunction() Set the function to be used for the get method of the radiativeEfficiency property of the BlackHoleStandard component.

<logical> radiativeEfficiencyIsAttached() Return true if the deferred function used to get the radiativeEfficiency property of the BlackHoleStandard component class has been attached.

<logical> radiativeEfficiencyIsGettable() Returns true if the radiativeEfficiency property is gettable for the blackHole component class.

<logical> radiativeEfficiencyIsSettable() Specify whether the radiativeEfficiency property of the blackHole component is settable.

<double precision> radiativeEfficiencyRateGet() Returns a zero rate for the radiativeEfficiency property for the blackHole component class.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a nonCentral implementation of the blackHole component.

<void> serializeASCII() Serialize the contents of a nonCentral implementation of the blackHole component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a nonCentral implementation of the blackHole component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a nonCentral implementation of the blackHole component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a nonCentral implementation of the blackHole component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a nonCentral implementation of the blackHole component to XML.

<logical> simpleIsActive() Return true if the simple implementation of the blackHole component is the active choice.

<integer(c_size_t)> sizeOf() Return the size in bytes of a nonCentral implementation of the blackHole component.

<double precision> spin() Returns the default value for the spin property for the blackHole component class.

<type(varying_string), allocatable, dimension(:) => matches> spinAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the blackHole class that have the desired attributes for the spin property

<integer> spinCount() Return a count of the number of scalar properties in the spin property of an standard implementation of the blackHole component class.

<void> spinInactive() Indicate that the spin property of an standard implementation of the blackHole component class is inactive for differential equation solving.

<logical> spinIsGettable() Returns true if the spin property is gettable for the blackHole component class.

<logical> spinIsSettable() Specify whether the spin property of the blackHole component is settable.

<void> spinRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the spin property of an standard implementation of the blackHole component class.

<double precision> spinRateGet() Get the rate of change of the spin property of an standard implementation of the blackHole component class.

<void> spinScale(<double precision> setValue→) Set the absolute scale of the spin property of an standard implementation of the blackHole component class.

<double precision> spinSeed() Returns the default value for the spinSeed property for the blackHole component class.

<type(varying_string), allocatable, dimension(:) => matches> spinSeedAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the blackHole class that have the desired attributes for the spinSeed property

<logical> spinSeedIsGettable() Returns true if the spinSeed property is gettable for the blackHole component class.

<logical> spinSeedIsSettable() Specify whether the spinSeed property of the blackHole component is settable.

<double precision> spinSeedRateGet() Returns a zero rate for the spinSeed property for the blackHole component class.

<void> spinSet(<double precision> setValue→) Set the spin property of an standard implementation of the blackHole component class.

<logical> standardIsActive() Return true if the standard implementation of the blackHole component is the active choice.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<double precision> tripleInteractionTime() Get the tripleInteractionTime property of an standard implementation of the blackHole component class.

<type(varying_string), allocatable, dimension(:) => matches> tripleInteractionTimeAttributeMatch(<logi [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the blackHole class that have the desired attributes for the tripleInteractionTime property

<logical> tripleInteractionTimeIsGettable() Returns true if the tripleInteractionTime property is gettable for the blackHole component class.

<logical> tripleInteractionTimeIsSettable() Specify whether the tripleInteractionTime property of the blackHole component is settable.

<double precision> tripleInteractionTimeRateGet() Returns a zero rate for the tripleInteractionTime property for the blackHole component class.

<void> tripleInteractionTimeSet(<double precision> setValue→) Set the tripleInteractionTime property of an standard implementation of the blackHole component class.

<type(varying_string)> type() Returns the type name for the nonCentral implementation of the blackHole component class.

nodeComponentBlackHoleNull

<double precision> accretionRate() Returns the default value for the accretionRate property for the blackHole component class.

<type(varying_string), allocatable, dimension(:) => matches> accretionRateAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the blackHole class that have the desired attributes for the accretionRate property

<logical> accretionRateIsGettable() Returns true if the accretionRate property is gettable for the blackHole component class.

<logical> accretionRateIsSettable() Specify whether the accretionRate property of the blackHole component is settable.

<double precision> accretionRateRateGet() Returns a zero rate for the accretionRate property for the blackHole component class.

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a null implementation of the blackHole component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a null implementation of the blackHole component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a null implementation of the blackHole component from array.

<void> destroy() Finalize a null implementation of the blackHole component.

<void> dumpASCII() Dump the content of a blackHole component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter
[interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a null member of the blackHole component.

<double precision> mass() Returns the default value for the mass property for the blackHole component class.

<type(varying_string), allocatable, dimension(:) => matches> massAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the blackHole class that have the desired attributes for the mass property

<integer> massCount() Compute the count of evolvable quantities in the mass property of the BlackHoleSimple component.

<logical> massIsGettable() Returns true if the mass property is gettable for the blackHole component class.

<logical> massIsSettable() Specify whether the mass property of the blackHole component is settable.

<void> massRate(<double precision> value) Cumulate to the rate of the mass property of the BlackHoleSimple component.

<double precision> massRateGet() Returns a zero rate for the mass property for the blackHole component class.

<void> massScale(<double precision> value) Set the scale of the mass property of the BlackHoleSimple component.

<double precision> massSeed() Returns the default value for the massSeed property for the blackHole component class.

<type(varying_string), allocatable, dimension(:) => matches> massSeedAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the blackHole class that have the desired attributes for the massSeed property

<logical> massSeedIsGettable() Returns true if the massSeed property is gettable for the blackHole component class.

<logical> massSeedIsSettable() Specify whether the massSeed property of the blackHole component is settable.

<double precision> massSeedRateGet() Returns a zero rate for the massSeed property for the blackHole component class.

<void> massSet(<double precision> value) Set the mass property of the blackHole component.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a null implementation of the blackHole component class.

<logical> nonCentralIsActive() Return true if the nonCentral implementation of the blackHole component is the active choice.

<logical> nullIsActive() Return true if the null implementation of the blackHole component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a blackHole component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a null implementation of the blackHole component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a null implementation of the blackHole component.

<void> postOutput(<double precision> time→) Perform post-output processing of a blackHole component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<double precision> radialPosition() Returns the default value for the radialPosition property for the blackHole component class.

<type(varying_string), allocatable, dimension(:) => matches> radialPositionAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the blackHole class that have the desired attributes for the radialPosition property

<integer> radialPositionCount() Compute the count of evolvable quantities in the radialPosition property of the BlackHoleNonCentral component.

<logical> radialPositionIsGettable() Returns true if the radialPosition property is gettable for the blackHole component class.

<logical> radialPositionIsSettable() Specify whether the radialPosition property of the blackHole component is settable.

<void> radialPositionRate(<double precision> value) Cumulate to the rate of the radialPosition property of the BlackHoleNonCentral component.

<double precision> radialPositionRateGet() Returns a zero rate for the radialPosition property for the blackHole component class.

<void> radialPositionScale(<double precision> value) Set the scale of the radialPosition property of the BlackHoleNonCentral component.

<void> radialPositionSet(<double precision> value) Set the radialPosition property of the blackHole component.

<double precision> radiativeEfficiency() Returns the default value for the radiativeEfficiency property for the blackHole component class.

<type(varying_string), allocatable, dimension(:) => matches> radiativeEfficiencyAttributeMatch(<logica
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the blackHole class that have the desired attributes for the radiativeEfficiency property

<logical> radiativeEfficiencyIsGettable() Returns true if the radiativeEfficiency property is gettable for the blackHole component class.

<logical> radiativeEfficiencyIsSettable() Specify whether the radiativeEfficiency property of the blackHole component is settable.

<double precision> radiativeEfficiencyRateGet() Returns a zero rate for the radiativeEfficiency property for the blackHole component class.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType>
[massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType>
[massType]→) Compute the rotation curve gradient.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→)
Compute offsets into serialization arrays for a null implementation of the blackHole component.

<void> serializeASCII() Serialize the contents of a null implementation of the blackHole component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a null implementation of the blackHole component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a null implementation of the blackHole component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a null implementation of the blackHole component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a null implementation of the blackHole component to XML.

<logical> simpleIsActive() Return true if the simple implementation of the blackHole component is the active choice.

<integer(c_size_t)> sizeOf() Return the size in bytes of a null implementation of the blackHole component.

<double precision> spin() Returns the default value for the spin property for the blackHole component class.

<type(varying_string), allocatable, dimension(:) => matches> spinAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the blackHole class that have the desired attributes for the spin property

<integer> spinCount() Compute the count of evolvable quantities in the spin property of the BlackHoleStandard component.

<logical> spinIsGettable() Returns true if the spin property is gettable for the blackHole component class.

<logical> spinIsSettable() Specify whether the spin property of the blackHole component is settable.

<void> spinRate(<double precision> value) Cumulate to the rate of the spin property of the BlackHoleStandard component.

<double precision> spinRateGet() Returns a zero rate for the spin property for the blackHole component class.

<void> spinScale(<double precision> value) Set the scale of the spin property of the BlackHoleStandard component.

<double precision> spinSeed() Returns the default value for the spinSeed property for the blackHole component class.

<type(varying_string), allocatable, dimension(:) => matches> spinSeedAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the blackHole class that have the desired attributes for the spinSeed property

<logical> spinSeedIsGettable() Returns true if the spinSeed property is gettable for the blackHole component class.

<logical> spinSeedIsSettable() Specify whether the spinSeed property of the blackHole component is settable.

<double precision> spinSeedRateGet() Returns a zero rate for the spinSeed property for the blackHole component class.

<void> spinSet(<double precision> value) Set the spin property of the blackHole component.

<logical> standardIsActive() Return true if the standard implementation of the blackHole component is the active choice.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<double precision> tripleInteractionTime() Returns the default value for the tripleInteractionTime property for the blackHole component class.

<type(varying_string), allocatable, dimension(:) => matches> tripleInteractionTimeAttributeMatch(<logi [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the blackHole class that have the desired attributes for the tripleInteractionTime property

<logical> tripleInteractionTimeIsGettable() Returns true if the tripleInteractionTime property is gettable for the blackHole component class.

<logical> tripleInteractionTimeIsSettable() Specify whether the tripleInteractionTime property of the blackHole component is settable.

<double precision> tripleInteractionTimeRateGet() Returns a zero rate for the tripleInteractionTime property for the blackHole component class.

<void> tripleInteractionTimeSet(<double precision> value) Set the tripleInteractionTime property of the blackHole component.

<type(varying_string)> type() Returns the type name for the null implementation of the blackHole component class.

nodeComponentBlackHoleSimple

<double precision> accretionRate() Returns the default value for the accretionRate property for the blackHole component class.

<type(varying_string), allocatable, dimension(:) => matches> accretionRateAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the blackHole class that have the desired attributes for the accretionRate property

<logical> accretionRateIsGettable() Returns true if the accretionRate property is gettable for the blackHole component class.

<logical> accretionRateIsSettable() Specify whether the accretionRate property of the blackHole component is settable.

<double precision> accretionRateRateGet() Returns a zero rate for the accretionRate property for the blackHole component class.

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a simple implementation of the blackHole component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a simple implementation of the blackHole component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a simple implementation of the blackHole component from array.

<void> destroy() Finalize a simple implementation of the blackHole component.

<void> dumpASCII() Dump the content of a blackHole component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a simple member of the blackHole component.

<double precision> mass() Get the mass property of an simple implementation of the blackHole component class.

<type(varying_string), allocatable, dimension(:) => matches> massAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the blackHole class that have the desired attributes for the mass property

<integer> massCount() Return a count of the number of scalar properties in the mass property of an simple implementation of the blackHole component class.

<void> massInactive() Indicate that the mass property of an simple implementation of the blackHole component class is inactive for differential equation solving.

<logical> massIsGettable() Returns true if the mass property is gettable for the blackHole component class.

<logical> massIsSettable() Specify whether the mass property of the blackHole component is settable.

<void> massRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the mass property of an simple implementation of the blackHole component class.

<double precision> massRateGet() Get the rate of change of the mass property of an simple implementation of the blackHole component class.

<void> massScale(<double precision> setValue→) Set the absolute scale of the mass property of an simple implementation of the blackHole component class.

<double precision> massSeed() Returns the default value for the massSeed property for the blackHole component class.

<type(varying_string), allocatable, dimension(:) => matches> massSeedAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the blackHole class that have the desired attributes for the massSeed property

<logical> massSeedIsGettable() Returns true if the massSeed property is gettable for the blackHole component class.

<logical> massSeedIsSettable() Specify whether the massSeed property of the blackHole component is settable.

<double precision> massSeedRateGet() Returns a zero rate for the massSeed property for the blackHole component class.

<void> massSet(<double precision> setValue→) Set the mass property of an simple implementation of the blackHole component class.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a simple implementation of the blackHole component class.

<logical> nonCentralIsActive() Return true if the nonCentral implementation of the blackHole component is the active choice.

<logical> nullIsActive() Return true if the null implementation of the blackHole component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a blackHole component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a simple implementation of the blackHole component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a simple implementation of the blackHole component.

<void> postOutput(<double precision> time→) Perform post-output processing of a blackHole component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<double precision> radialPosition() Returns the default value for the radialPosition property for the blackHole component class.

<type(varying_string), allocatable, dimension(:) => matches> radialPositionAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the blackHole class that have the desired attributes for the radialPosition property

`<integer> radialPositionCount()` Compute the count of evolvable quantities in the `radialPosition` property of the `BlackHoleNonCentral` component.

`<logical> radialPositionIsGettable()` Returns true if the `radialPosition` property is gettable for the `blackHole` component class.

`<logical> radialPositionIsSettable()` Specify whether the `radialPosition` property of the `blackHole` component is settable.

`<void> radialPositionRate(<double precision> value)` Cumulate to the rate of the `radialPosition` property of the `BlackHoleNonCentral` component.

`<double precision> radialPositionRateGet()` Returns a zero rate for the `radialPosition` property for the `blackHole` component class.

`<void> radialPositionScale(<double precision> value)` Set the scale of the `radialPosition` property of the `BlackHoleNonCentral` component.

`<void> radialPositionSet(<double precision> value)` Set the `radialPosition` property of the `blackHole` component.

`<double precision> radiativeEfficiency()` Returns the default value for the `radiativeEfficiency` property for the `blackHole` component class.

`<type(varying_string), allocatable, dimension(:) => matches> radiativeEfficiencyAttributeMatch(<logica [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)` Return a text list of component implementations in the `blackHole` class that have the desired attributes for the `radiativeEfficiency` property

`<logical> radiativeEfficiencyIsGettable()` Returns true if the `radiativeEfficiency` property is gettable for the `blackHole` component class.

`<logical> radiativeEfficiencyIsSettable()` Specify whether the `radiativeEfficiency` property of the `blackHole` component is settable.

`<double precision> radiativeEfficiencyRateGet()` Returns a zero rate for the `radiativeEfficiency` property for the `blackHole` component class.

`<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→)` Compute the rotation curve.

`<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→)` Compute the rotation curve gradient.

`<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→)` Compute offsets into serialization arrays for a `simple` implementation of the `blackHole` component.

`<void> serializeASCII()` Serialize the contents of a simple implementation of the blackHole component to ASCII.

`<integer> serializeCount(<integer> propertyType→)` Return a count of the serialization of a simple implementation of the blackHole component.

`<void> serializeRaw(<integer> fileHandle→)` Serialize the contents of a simple implementation of the blackHole component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a simple implementation of the blackHole component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a simple implementation of the blackHole component to XML.

<logical> simpleIsActive() Return true if the simple implementation of the blackHole component is the active choice.

<integer(c_size_t)> sizeOf() Return the size in bytes of a simple implementation of the blackHole component.

<double precision> spin() Returns the default value for the spin property for the blackHole component class.

<type(varying_string), allocatable, dimension(:) => matches> spinAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the blackHole class that have the desired attributes for the spin property

<integer> spinCount() Compute the count of evolvable quantities in the spin property of the BlackHoleStandard component.

<logical> spinIsGettable() Returns true if the spin property is gettable for the blackHole component class.

<logical> spinIsSettable() Specify whether the spin property of the blackHole component is settable.

<void> spinRate(<double precision> value) Cumulate to the rate of the spin property of the BlackHoleStandard component.

<double precision> spinRateGet() Returns a zero rate for the spin property for the blackHole component class.

<void> spinScale(<double precision> value) Set the scale of the spin property of the BlackHoleStandard component.

<double precision> spinSeed() Returns the default value for the spinSeed property for the blackHole component class.

<type(varying_string), allocatable, dimension(:) => matches> spinSeedAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the blackHole class that have the desired attributes for the spinSeed property

<logical> spinSeedIsGettable() Returns true if the spinSeed property is gettable for the blackHole component class.

<logical> spinSeedIsSettable() Specify whether the spinSeed property of the blackHole component is settable.

<double precision> spinSeedRateGet() Returns a zero rate for the spinSeed property for the blackHole component class.

<void> spinSet(<double precision> value) Set the spin property of the blackHole component.

<logical> standardIsActive() Return true if the standard implementation of the blackHole component is the active choice.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<double precision> tripleInteractionTime() Returns the default value for the tripleInteractionTime property for the blackHole component class.

<type(varying_string), allocatable, dimension(:) => matches> tripleInteractionTimeAttributeMatch(<logi [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the blackHole class that have the desired attributes for the tripleInteractionTime property

<logical> tripleInteractionTimeIsGettable() Returns true if the tripleInteractionTime property is gettable for the blackHole component class.

<logical> tripleInteractionTimeIsSettable() Specify whether the tripleInteractionTime property of the blackHole component is settable.

<double precision> tripleInteractionTimeRateGet() Returns a zero rate for the tripleInteractionTime property for the blackHole component class.

<void> tripleInteractionTimeSet(<double precision> value) Set the tripleInteractionTime property of the blackHole component.

<type(varying_string)> type() Returns the type name for the simple implementation of the blackHole component class.

nodeComponentBlackHoleStandard

<double precision> accretionRate() Get the value of the accretionRate property of the standard implementation of the blackHole component using a deferred function.

<type(varying_string), allocatable, dimension(:) => matches> accretionRateAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the blackHole class that have the desired attributes for the accretionRate property

<void> accretionRateFunction() Set the function to be used for the get method of the accretionRate property of the BlackHoleStandard component.

<logical> accretionRateIsAttached() Return true if the deferred function used to get the accretionRate property of the BlackHoleStandard component class has been attached.

<logical> accretionRateIsGettable() Returns true if the accretionRate property is gettable for the blackHole component class.

<logical> accretionRateIsSettable() Specify whether the accretionRate property of the blackHole component is settable.

<double precision> accretionRateRateGet() Returns a zero rate for the accretionRate property for the blackHole component class.

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a standard implementation of the blackHole component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a standard implementation of the blackHole component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a standard implementation of the blackHole component from array.

<void> destroy() Finalize a standard implementation of the blackHole component.

<void> dumpASCII() Dump the content of a blackHole component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a standard member of the blackHole component.

<double precision> mass() Get the mass property of an standard implementation of the blackHole component class.

<type(varying_string), allocatable, dimension(:) => matches> massAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the blackHole class that have the desired attributes for the mass property

<integer> massCount() Return a count of the number of scalar properties in the mass property of an standard implementation of the blackHole component class.

<void> massInactive() Indicate that the mass property of an standard implementation of the blackHole component class is inactive for differential equation solving.

<logical> massIsGettable() Returns true if the mass property is gettable for the blackHole component class.

<logical> massIsSettable() Specify whether the mass property of the blackHole component is settable.

<void> massRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the mass property of an standard implementation of the blackHole component class.

<double precision> massRateGet() Get the rate of change of the mass property of an standard implementation of the blackHole component class.

<void> massScale(<double precision> setValue→) Set the absolute scale of the mass property of an standard implementation of the blackHole component class.

<double precision> massSeed() Returns the default value for the massSeed property for the blackHole component class.

<type(varying_string), allocatable, dimension(:) => matches> massSeedAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the blackHole class that have the desired attributes for the massSeed property

<logical> massSeedIsGettable() Returns true if the massSeed property is gettable for the blackHole component class.

<logical> massSeedIsSettable() Specify whether the massSeed property of the blackHole component is settable.

<double precision> massSeedRateGet() Returns a zero rate for the massSeed property for the blackHole component class.

<void> massSet(<double precision> setValue→) Set the mass property of an standard implementation of the blackHole component class.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a standard implementation of the blackHole component class.

<logical> nonCentralIsActive() Return true if the nonCentral implementation of the blackHole component is the active choice.

<logical> nullIsActive() Return true if the null implementation of the blackHole component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a blackHole component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a standard implementation of the blackHole component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a standard implementation of the blackHole component.

<void> postOutput(<double precision> time→) Perform post-output processing of a blackHole component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<double precision> radialPosition() Get the radialPosition property of an standard implementation of the blackHole component class.

<type(varying_string), allocatable, dimension(:) => matches> radialPositionAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the blackHole class that have the desired attributes for the radialPosition property

<integer> radialPositionCount() Compute the count of evolvable quantities in the radialPosition property of the BlackHoleNonCentral component.

<logical> radialPositionIsGettable() Returns true if the radialPosition property is gettable for the blackHole component class.

<logical> radialPositionIsSettable() Specify whether the radialPosition property of the blackHole component is settable.

<void> radialPositionRate(<double precision> value) Cumulate to the rate of the radialPosition property of the BlackHoleNonCentral component.

<double precision> radialPositionRateGet() Returns a zero rate for the radialPosition property for the blackHole component class.

<void> radialPositionScale(<double precision> value) Set the scale of the radialPosition property of the BlackHoleNonCentral component.

<void> radialPositionSet(<double precision> setValue→) Set the radialPosition property of an standard implementation of the blackHole component class.

<double precision> radiativeEfficiency() Get the value of the radiativeEfficiency property of the standard implementation of the blackHole component using a deferred function.

<type(varying_string), allocatable, dimension(:) => matches> radiativeEfficiencyAttributeMatch(<logica [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the blackHole class that have the desired attributes for the radiativeEfficiency property

<void> radiativeEfficiencyFunction() Set the function to be used for the get method of the radiativeEfficiency property of the BlackHoleStandard component.

<logical> radiativeEfficiencyIsAttached() Return true if the deferred function used to get the radiativeEfficiency property of the BlackHoleStandard component class has been attached.

<logical> radiativeEfficiencyIsGettable() Returns true if the radiativeEfficiency property is gettable for the blackHole component class.

<logical> radiativeEfficiencyIsSettable() Specify whether the radiativeEfficiency property of the blackHole component is settable.

<double precision> radiativeEfficiencyRateGet() Returns a zero rate for the radiativeEfficiency property for the blackHole component class.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a standard implementation of the blackHole component.

<void> serializeASCII() Serialize the contents of a standard implementation of the blackHole component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a standard implementation of the blackHole component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a standard implementation of the blackHole component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a standard implementation of the blackHole component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a standard implementation of the blackHole component to XML.

<logical> simpleIsActive() Return true if the simple implementation of the blackHole component is the active choice.

<integer(c_size_t)> sizeOf() Return the size in bytes of a standard implementation of the blackHole component.

<double precision> spin() Returns the default value for the spin property for the blackHole component class.

<type(varying_string), allocatable, dimension(:) => matches> spinAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the blackHole class that have the desired attributes for the spin property

<integer> spinCount() Return a count of the number of scalar properties in the spin property of an standard implementation of the blackHole component class.

<void> spinInactive() Indicate that the spin property of an standard implementation of the blackHole component class is inactive for differential equation solving.

<logical> spinIsGettable() Returns true if the spin property is gettable for the blackHole component class.

<logical> spinIsSettable() Specify whether the spin property of the blackHole component is settable.

<void> spinRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the spin property of an standard implementation of the blackHole component class.

<double precision> spinRateGet() Get the rate of change of the spin property of an standard implementation of the blackHole component class.

<void> spinScale(<double precision> setValue→) Set the absolute scale of the spin property of an standard implementation of the blackHole component class.

<double precision> spinSeed() Returns the default value for the spinSeed property for the blackHole component class.

<type(varying_string), allocatable, dimension(:) => matches> spinSeedAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the blackHole class that have the desired attributes for the spinSeed property

<logical> spinSeedIsGettable() Returns true if the spinSeed property is gettable for the blackHole component class.

<logical> spinSeedIsSettable() Specify whether the spinSeed property of the blackHole component is settable.

<double precision> spinSeedRateGet() Returns a zero rate for the spinSeed property for the blackHole component class.

<void> spinSet(<double precision> setValue→) Set the spin property of an standard implementation of the blackHole component class.

<logical> standardIsActive() Return true if the standard implementation of the blackHole component is the active choice.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<double precision> tripleInteractionTime() Get the tripleInteractionTime property of an standard implementation of the blackHole component class.

<type(varying_string), allocatable, dimension(:) => matches> tripleInteractionTimeAttributeMatch(<logi [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the blackHole class that have the desired attributes for the tripleInteractionTime property

<logical> tripleInteractionTimeIsGettable() Returns true if the tripleInteractionTime property is gettable for the blackHole component class.

<logical> tripleInteractionTimeIsSettable() Specify whether the tripleInteractionTime property of the blackHole component is settable.

<double precision> tripleInteractionTimeRateGet() Returns a zero rate for the tripleInteractionTime property for the blackHole component class.

<void> tripleInteractionTimeSet(<double precision> setValue→) Set the tripleInteractionTime property of an standard implementation of the blackHole component class.

<type(varying_string)> type() Returns the type name for the standard implementation of the blackHole component class.

nodeComponentDarkMatterProfile

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a generic darkMatterProfile component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Read properties from raw file.

<void> deserializeValues(<double(:)> array→, <integer> propertyType→) Deserialize the evolvable quantities from an array.

<void> destroy() Finalize a generic darkMatterProfile component.

<void> dumpASCII() Dump the content of a darkMatterProfile component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a generic darkMatterProfile component.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a nodeComponent object.

<logical> nullIsActive() Return true if the null implementation of the darkMatterProfile component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a darkMatterProfile component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a generic darkMatterProfile component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Establish the names of properties to output for a generic darkMatterProfile component.

<void> postOutput(<double precision> time→) Perform post-output processing of a darkMatterProfile component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<double precision> scale() Returns the default value for the scale property for the darkMatterProfile component class.

<type(varying_string), allocatable, dimension(:) => matches> scaleAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the darkMatterProfile class that have the desired attributes for the scale property

<integer> scaleCount() Compute the count of evolvable quantities in the scale property of the DarkMatterProfileScale component.

<double precision> scaleGrowthRate() Returns the default value for the scaleGrowthRate property for the darkMatterProfile component class.

<type(varying_string), allocatable, dimension(:) => matches> scaleGrowthRateAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the darkMatterProfile class that have the desired attributes for the scaleGrowthRate property

<logical> scaleGrowthRateIsGettable() Returns true if the scaleGrowthRate property is gettable for the darkMatterProfile component class.

<logical> scaleGrowthRateIsSettable() Specify whether the scaleGrowthRate property of the darkMatterProfile component is settable.

<double precision> scaleGrowthRateRateGet() Returns a zero rate for the scaleGrowthRate property for the darkMatterProfile component class.

<void> scaleGrowthRateSet(<double precision> value) Set the scaleGrowthRate property of the darkMatterProfile component.

<logical> scaleIsActive() Return true if the scale implementation of the darkMatterProfile component is the active choice.

<logical> scaleIsGettable() Returns true if the scale property is gettable for the darkMatterProfile component class.

<logical> scaleIsLimited() Returns the default value for the scaleIsLimited property for the darkMatterProfile component class.

<type(varying_string), allocatable, dimension(:) => matches> scaleIsLimitedAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the darkMatterProfile class that have the desired attributes for the scaleIsLimited property

<logical> scaleIsLimitedIsGettable() Returns true if the scaleIsLimited property is gettable for the darkMatterProfile component class.

<logical> scaleIsLimitedIsSettable() Specify whether the scaleIsLimited property of the darkMatterProfile component is settable.

<logical> scaleIsLimitedRateGet() Returns a zero rate for the scaleIsLimited property for the darkMatterProfile component class.

<void> scaleIsLimitedSet(<logical> value) Set the scaleIsLimited property of the darkMatterProfile component.

<logical> scaleIsSettable() Specify whether the scale property of the darkMatterProfile component is settable.

<logical> scalePresetIsActive() Return true if the scalePreset implementation of the darkMatterProfile component is the active choice.

<void> scaleRate(<double precision> value) Cumulate to the rate of the scale property of the DarkMatterProfileScale component.

<double precision> scaleRateGet() Returns a zero rate for the scale property for the darkMatterProfile component class.

<void> scaleScale(<double precision> value) Set the scale of the scale property of the DarkMatterProfileScale component.

<void> scaleSet(<double precision> value) Set the scale property of the darkMatterProfile component.

<logical> scaleShapeIsActive() Return true if the scaleShape implementation of the darkMatterProfile component is the active choice.

<void> serializationOffsets() Set offsets into serialization arrays.

<void> serializeASCII() Serialize the content of a darkMatterProfile component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the number of evolvable quantities to be evolved.

<void> serializeRaw(<integer> fileHandle→) Generate a binary dump of all properties.

<void> serializeValues(<double(:)> array←, <integer> propertyType→) Serialize the evolvable quantities to an array.

<void> serializeXML() Generate an XML dump of all properties.

<double precision> shape() Returns the default value for the shape property for the darkMatterProfile component class.

<type(varying_string), allocatable, dimension(:) => matches> shapeAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the darkMatterProfile class that have the desired attributes for the shape property

<integer> shapeCount() Compute the count of evolvable quantities in the shape property of the DarkMatterProfileScaleShape component.

<double precision> shapeGrowthRate() Returns the default value for the shapeGrowthRate property for the darkMatterProfile component class.

<type(varying_string), allocatable, dimension(:) => matches> shapeGrowthRateAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the darkMatterProfile class that have the desired attributes for the shapeGrowthRate property

<logical> shapeGrowthRateIsGettable() Returns true if the shapeGrowthRate property is gettable for the darkMatterProfile component class.

<logical> shapeGrowthRateIsSettable() Specify whether the shapeGrowthRate property of the darkMatterProfile component is settable.

<double precision> shapeGrowthRateRateGet() Returns a zero rate for the shapeGrowthRate property for the darkMatterProfile component class.

<void> shapeGrowthRateSet(<double precision> value) Set the shapeGrowthRate property of the darkMatterProfile component.

<logical> shapeIsGettable() Returns true if the shape property is gettable for the darkMatterProfile component class.

<logical> shapeIsSettable() Specify whether the shape property of the darkMatterProfile component is settable.

<void> shapeRate(<double precision> value) Cumulate to the rate of the shape property of the DarkMatterProfileScaleShape component.

<double precision> shapeRateGet() Returns a zero rate for the shape property for the darkMatterProfile component class.

<void> shapeScale(<double precision> value) Set the scale of the shape property of the DarkMatterProfileScaleShap component.

<void> shapeSet(<double precision> value) Set the shape property of the darkMatterProfile component.

<integer(c_size_t)> sizeOf() Return the size in bytes of a nodeComponentDarkMatterProfile component.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<type(varying_string)> type() Returns the type name for the darkMatterProfile component class.

nodeComponentDarkMatterProfileNull

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a null implementation of the darkMatterProfile component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a null implementation of the darkMatterProfile component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a null implementation of the darkMatterProfile component from array.

<void> destroy() Finalize a null implementation of the darkMatterProfile component.

<void> dumpASCII() Dump the content of a darkMatterProfile component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter
[interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a null member of the darkMatterProfile component.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a null implementation of the darkMatterProfile component class.

<logical> nullIsActive() Return true if the null implementation of the darkMatterProfile component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_-
int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔,
<double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)>
outputInstance→, <integer> instance→) Populate output buffers with properties to output for a darkMatterProfile component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double
precision> time→, <integer> instance→) Increment the count of properties to output for a null implementation of the darkMatterProfile component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔,
<character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔,
<integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]>
doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision>
time→, <integer> instance→) Return the names of properties to output for a null implementation of the darkMatterProfile component.

<void> postOutput(<double precision> time→) Perform post-output processing of a darkMatterProfile component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→)
Compute the gravitational potential.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType>
[massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType>
[massType]→) Compute the rotation curve gradient.

<double precision> scale() Returns the default value for the scale property for the darkMatterProfile component class.

<type(varying_string), allocatable, dimension(:) => matches> scaleAttributeMatch(<logical>
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the darkMatterProfile class that have the desired attributes for the scale property

<integer> scaleCount() Compute the count of evolvable quantities in the scale property of the DarkMatterProfileScale component.

<double precision> scaleGrowthRate() Returns the default value for the scaleGrowthRate property for the darkMatterProfile component class.

<type(varying_string), allocatable, dimension(:) => matches> scaleGrowthRateAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the darkMatterProfile class that have the desired attributes for the scaleGrowthRate property

<logical> scaleGrowthRateIsGettable() Returns true if the scaleGrowthRate property is gettable for the darkMatterProfile component class.

<logical> scaleGrowthRateIsSettable() Specify whether the scaleGrowthRate property of the darkMatterProfile component is settable.

<double precision> scaleGrowthRateRateGet() Returns a zero rate for the scaleGrowthRate property for the darkMatterProfile component class.

<void> scaleGrowthRateSet(<double precision> value) Set the scaleGrowthRate property of the darkMatterProfile component.

<logical> scaleIsActive() Return true if the scale implementation of the darkMatterProfile component is the active choice.

<logical> scaleIsGettable() Returns true if the scale property is gettable for the darkMatterProfile component class.

<logical> scaleIsLimited() Returns the default value for the scaleIsLimited property for the darkMatterProfile component class.

<type(varying_string), allocatable, dimension(:) => matches> scaleIsLimitedAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the darkMatterProfile class that have the desired attributes for the scaleIsLimited property

<logical> scaleIsLimitedIsGettable() Returns true if the scaleIsLimited property is gettable for the darkMatterProfile component class.

<logical> scaleIsLimitedIsSettable() Specify whether the scaleIsLimited property of the darkMatterProfile component is settable.

<logical> scaleIsLimitedRateGet() Returns a zero rate for the scaleIsLimited property for the darkMatterProfile component class.

<void> scaleIsLimitedSet(<logical> value) Set the scaleIsLimited property of the darkMatterProfile component.

<logical> scaleIsSettable() Specify whether the scale property of the darkMatterProfile component is settable.

<logical> scalePresetIsActive() Return true if the scalePreset implementation of the darkMatterProfile component is the active choice.

<void> scaleRate(<double precision> value) Cumulate to the rate of the scale property of the DarkMatterProfileScale component.

<double precision> scaleRateGet() Returns a zero rate for the scale property for the darkMatterProfile component class.

<void> scaleScale(<double precision> value) Set the scale of the scale property of the DarkMatterProfileScale component.

<void> scaleSet(<double precision> value) Set the scale property of the darkMatterProfile component.

<logical> scaleShapeIsActive() Return true if the scaleShape implementation of the darkMatterProfile component is the active choice.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a null implementation of the darkMatterProfile component.

<void> serializeASCII() Serialize the contents of a null implementation of the darkMatterProfile component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a null implementation of the darkMatterProfile component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a null implementation of the darkMatterProfile component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a null implementation of the darkMatterProfile component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a null implementation of the darkMatterProfile component to XML.

<double precision> shape() Returns the default value for the shape property for the darkMatterProfile component class.

<type(varying_string), allocatable, dimension(:) => matches> shapeAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the darkMatterProfile class that have the desired attributes for the shape property

<integer> shapeCount() Compute the count of evolvable quantities in the shape property of the DarkMatterProfileScaleShape component.

<double precision> shapeGrowthRate() Returns the default value for the shapeGrowthRate property for the darkMatterProfile component class.

<type(varying_string), allocatable, dimension(:) => matches> shapeGrowthRateAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the darkMatterProfile class that have the desired attributes for the shapeGrowthRate property

<logical> shapeGrowthRateIsGettable() Returns true if the shapeGrowthRate property is gettable for the darkMatterProfile component class.

<logical> shapeGrowthRateIsSettable() Specify whether the shapeGrowthRate property of the darkMatterProfile component is settable.

<double precision> shapeGrowthRateRateGet() Returns a zero rate for the shapeGrowthRate property for the darkMatterProfile component class.

<void> shapeGrowthRateSet(<double precision> value) Set the shapeGrowthRate property of the darkMatterProfile component.

<logical> shapeIsGettable() Returns true if the shape property is gettable for the darkMatterProfile component class.

<logical> shapeIsSettable() Specify whether the shape property of the darkMatterProfile component is settable.

<void> shapeRate(<double precision> value) Cumulate to the rate of the shape property of the DarkMatterProfileScaleShape component.

<double precision> shapeRateGet() Returns a zero rate for the shape property for the darkMatterProfile component class.

<void> shapeScale(<double precision> value) Set the scale of the shape property of the DarkMatterProfileScaleShap component.

<void> shapeSet(<double precision> value) Set the shape property of the darkMatterProfile component.

<integer(c_size_t)> sizeOf() Return the size in bytes of a null implementation of the darkMatterProfile component.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<type(varying_string)> type() Returns the type name for the null implementation of the darkMatterProfile component class.

nodeComponentDarkMatterProfileScale

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a scale implementation of the darkMatterProfile component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a scale implementation of the darkMatterProfile component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a scale implementation of the darkMatterProfile component from array.

<void> destroy() Finalize a scale implementation of the darkMatterProfile component.

<void> dumpASCII() Dump the content of a darkMatterProfile component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a scale member of the darkMatterProfile component.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a scale implementation of the darkMatterProfile component class.

<logical> nullIsActive() Return true if the null implementation of the darkMatterProfile component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a darkMatterProfile component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a scale implementation of the darkMatterProfile component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a scale implementation of the darkMatterProfile component.

<void> postOutput(<double precision> time→) Perform post-output processing of a darkMatterProfile component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<double precision> scale() Get the value of the scale property of the scale implementation of the darkMatterProfile component using a deferred function.

<type(varying_string), allocatable, dimension(:) => matches> scaleAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the darkMatterProfile class that have the desired attributes for the scale property

<integer> scaleCount() Return a count of the number of scalar properties in the scale property of an scale implementation of the darkMatterProfile component class.

<void> scaleFunction() Set the function to be used for the get method of the scale property of the DarkMatterProfileScale component.

<double precision> scaleGrowthRate() Get the scaleGrowthRate property of an scale implementation of the darkMatterProfile component class.

<type(varying_string), allocatable, dimension(:) => matches> scaleGrowthRateAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the darkMatterProfile class that have the desired attributes for the scaleGrowthRate property

<logical> scaleGrowthRateIsGettable() Returns true if the scaleGrowthRate property is gettable for the darkMatterProfile component class.

<logical> scaleGrowthRateIsSettable() Specify whether the scaleGrowthRate property of the darkMatterProfile component is settable.

<double precision> scaleGrowthRateRateGet() Returns a zero rate for the scaleGrowthRate property for the darkMatterProfile component class.

<void> scaleGrowthRateSet(<double precision> setValue→) Set the scaleGrowthRate property of an scale implementation of the darkMatterProfile component class.

<void> scaleInactive() Indicate that the scale property of an scale implementation of the darkMatterProfile component class is inactive for differential equation solving.

<logical> scaleIsActive() Return true if the scale implementation of the darkMatterProfile component is the active choice.

<logical> scaleIsAttached() Return true if the deferred function used to get the scale property of the DarkMatterProfileScale component class has been attached.

<logical> scaleIsGettable() Returns true if the scale property is gettable for the darkMatterProfile component class.

<logical> scaleIsLimited() Get the scaleIsLimited property of an scale implementation of the darkMatterProfile component class.

<type(varying_string), allocatable, dimension(:) => matches> scaleIsLimitedAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the darkMatterProfile class that have the desired attributes for the scaleIsLimited property

<logical> scaleIsLimitedIsGettable() Returns true if the scaleIsLimited property is gettable for the darkMatterProfile component class.

<logical> scaleIsLimitedIsSettable() Specify whether the scaleIsLimited property of the darkMatterProfile component is settable.

<logical> scaleIsLimitedRateGet() Returns a zero rate for the scaleIsLimited property for the darkMatterProfile component class.

<void> scaleIsLimitedSet(<logical> setValue→) Set the scaleIsLimited property of an scale implementation of the darkMatterProfile component class.

<logical> scaleIsSettable() Specify whether the scale property of the darkMatterProfile component is settable.

<logical> scalePresetIsActive() Return true if the scalePreset implementation of the darkMatterProfile component is the active choice.

<void> scaleRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the scale property of an scale implementation of the darkMatterProfile component class.

<double precision> scaleRateGet() Get the rate of change of the scale property of an scale implementation of the darkMatterProfile component class.

<void> scaleScale(<double precision> setValue→) Set the absolute scale of the scale property of an scale implementation of the darkMatterProfile component class.

<void> scaleSet(<double precision> setValue→) Set the scale property of an scale implementation of the darkMatterProfile component class.

<logical> scaleShapeIsActive() Return true if the scaleShape implementation of the darkMatterProfile component is the active choice.

<double precision> scaleValue() Get the scale property of an scale implementation of the darkMatterProfile component class.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a scale implementation of the darkMatterProfile component.

<void> serializeASCII() Serialize the contents of a scale implementation of the darkMatterProfile component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a scale implementation of the darkMatterProfile component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a scale implementation of the darkMatterProfile component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a scale implementation of the darkMatterProfile component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a scale implementation of the darkMatterProfile component to XML.

<double precision> shape() Returns the default value for the shape property for the darkMatterProfile component class.

<type(varying_string), allocatable, dimension(:) => matches> shapeAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the darkMatterProfile class that have the desired attributes for the shape property

<integer> shapeCount() Compute the count of evolvable quantities in the shape property of the DarkMatterProfileScaleShape component.

<double precision> shapeGrowthRate() Returns the default value for the shapeGrowthRate property for the darkMatterProfile component class.

<type(varying_string), allocatable, dimension(:) => matches> shapeGrowthRateAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the darkMatterProfile class that have the desired attributes for the shapeGrowthRate property

<logical> shapeGrowthRateIsGettable() Returns true if the shapeGrowthRate property is gettable for the darkMatterProfile component class.

<logical> shapeGrowthRateIsSettable() Specify whether the shapeGrowthRate property of the darkMatterProfile component is settable.

<double precision> shapeGrowthRateRateGet() Returns a zero rate for the shapeGrowthRate property for the darkMatterProfile component class.

<void> shapeGrowthRateSet(<double precision> value) Set the shapeGrowthRate property of the darkMatterProfile component.

<logical> shapeIsGettable() Returns true if the shape property is gettable for the darkMatterProfile component class.

<logical> shapeIsSettable() Specify whether the shape property of the darkMatterProfile component is settable.

<void> shapeRate(<double precision> value) Cumulate to the rate of the shape property of the DarkMatterProfileScaleShape component.

<double precision> shapeRateGet() Returns a zero rate for the shape property for the darkMatterProfile component class.

<void> shapeScale(<double precision> value) Set the scale of the shape property of the DarkMatterProfileScaleShap component.

<void> shapeSet(<double precision> value) Set the shape property of the darkMatterProfile component.

<integer(c_size_t)> sizeOf() Return the size in bytes of a scale implementation of the darkMatterProfile component.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<type(varying_string)> type() Returns the type name for the scale implementation of the darkMatterProfile component class.

nodeComponentDarkMatterProfileScalePreset

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a scalePreset implementation of the darkMatterProfile component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a scalePreset implementation of the darkMatterProfile component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a scalePreset implementation of the darkMatterProfile component from array.

<void> destroy() Finalize a scalePreset implementation of the darkMatterProfile component.

<void> dumpASCII() Dump the content of a darkMatterProfile component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a scalePreset member of the darkMatterProfile component.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a scalePreset implementation of the darkMatterProfile component class.

<logical> nullIsActive() Return true if the null implementation of the darkMatterProfile component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a darkMatterProfile component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a scalePreset implementation of the darkMatterProfile component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a scalePreset implementation of the darkMatterProfile component.

<void> postOutput(<double precision> time→) Perform post-output processing of a darkMatterProfile component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<double precision> scale() Get the scale property of an scalePreset implementation of the darkMatterProfile component class.

<type(varying_string), allocatable, dimension(:) => matches> scaleAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the darkMatterProfile class that have the desired attributes for the scale property

<integer> scaleCount() Return a count of the number of scalar properties in the scale property of an scalePreset implementation of the darkMatterProfile component class.

<double precision> scaleGrowthRate() Get the scaleGrowthRate property of an scalePreset implementation of the darkMatterProfile component class.

<type(varying_string), allocatable, dimension(:) => matches> scaleGrowthRateAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the darkMatterProfile class that have the desired attributes for the scaleGrowthRate property

<logical> scaleGrowthRateIsGettable() Returns true if the scaleGrowthRate property is gettable for the darkMatterProfile component class.

<logical> scaleGrowthRateIsSettable() Specify whether the scaleGrowthRate property of the darkMatterProfile component is settable.

<double precision> scaleGrowthRateRateGet() Returns a zero rate for the scaleGrowthRate property for the darkMatterProfile component class.

<void> scaleGrowthRateSet(<double precision> setValue→) Set the scaleGrowthRate property of an scalePreset implementation of the darkMatterProfile component class.

<void> scaleInactive() Indicate that the scale property of an scalePreset implementation of the darkMatterProfile component class is inactive for differential equation solving.

<logical> scaleIsActive() Return true if the scale implementation of the darkMatterProfile component is the active choice.

<logical> scaleIsGettable() Returns true if the scale property is gettable for the darkMatterProfile component class.

<logical> scaleIsLimited() Returns the default value for the scaleIsLimited property for the darkMatterProfile component class.

<type(varying_string), allocatable, dimension(:) => matches> scaleIsLimitedAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the darkMatterProfile class that have the desired attributes for the scaleIsLimited property

<logical> scaleIsLimitedIsGettable() Returns true if the scaleIsLimited property is gettable for the darkMatterProfile component class.

<logical> scaleIsLimitedIsSettable() Specify whether the scaleIsLimited property of the darkMatterProfile component is settable.

<logical> scaleIsLimitedRateGet() Returns a zero rate for the scaleIsLimited property for the darkMatterProfile component class.

<void> scaleIsLimitedSet(<logical> value) Set the scaleIsLimited property of the darkMatterProfile component.

<logical> scaleIsSettable() Specify whether the scale property of the darkMatterProfile component is settable.

<logical> scalePresetIsActive() Return true if the scalePreset implementation of the darkMatterProfile component is the active choice.

<void> scaleRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the scale property of an scalePreset implementation of the darkMatterProfile component class.

<double precision> scaleRateGet() Get the rate of change of the scale property of an scalePreset implementation of the darkMatterProfile component class.

<void> scaleScale(<double precision> setValue→) Set the absolute scale of the scale property of an scalePreset implementation of the darkMatterProfile component class.

<void> scaleSet(<double precision> setValue→) Set the scale property of an scalePreset implementation of the darkMatterProfile component class.

<logical> scaleShapeIsActive() Return true if the scaleShape implementation of the darkMatterProfile component is the active choice.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a scalePreset implementation of the darkMatterProfile component.

<void> serializeASCII() Serialize the contents of a scalePreset implementation of the darkMatterProfile component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a scalePreset implementation of the darkMatterProfile component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a scalePreset implementation of the darkMatterProfile component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a scalePreset implementation of the darkMatterProfile component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a scalePreset implementation of the darkMatterProfile component to XML.

<double precision> shape() Returns the default value for the shape property for the darkMatterProfile component class.

<type(varying_string), allocatable, dimension(:) => matches> shapeAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the darkMatterProfile class that have the desired attributes for the shape property

<integer> shapeCount() Compute the count of evolvable quantities in the shape property of the DarkMatterProfileScaleShape component.

<double precision> shapeGrowthRate() Returns the default value for the shapeGrowthRate property for the darkMatterProfile component class.

<type(varying_string), allocatable, dimension(:) => matches> shapeGrowthRateAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the darkMatterProfile class that have the desired attributes for the shapeGrowthRate property

<logical> shapeGrowthRateIsGettable() Returns true if the shapeGrowthRate property is gettable for the darkMatterProfile component class.

<logical> shapeGrowthRateIsSettable() Specify whether the shapeGrowthRate property of the darkMatterProfile component is settable.

<double precision> shapeGrowthRateRateGet() Returns a zero rate for the shapeGrowthRate property for the darkMatterProfile component class.

<void> shapeGrowthRateSet(<double precision> value) Set the shapeGrowthRate property of the darkMatterProfile component.

<logical> shapeIsGettable() Returns true if the shape property is gettable for the darkMatterProfile component class.

<logical> shapeIsSettable() Specify whether the shape property of the darkMatterProfile component is settable.

<void> shapeRate(<double precision> value) Cumulate to the rate of the shape property of the DarkMatterProfileScaleShape component.

<double precision> shapeRateGet() Returns a zero rate for the shape property for the darkMatterProfile component class.

<void> shapeScale(<double precision> value) Set the scale of the shape property of the DarkMatterProfileScaleShap component.

<void> shapeSet(<double precision> value) Set the shape property of the darkMatterProfile component.

<integer(c_size_t)> sizeOf() Return the size in bytes of a scalePreset implementation of the darkMatterProfile component.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<type(varying_string)> type() Returns the type name for the scalePreset implementation of the darkMatterProfile component class.

nodeComponentDarkMatterProfileScaleShape

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a scaleShape implementation of the darkMatterProfile component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a scaleShape implementation of the darkMatterProfile component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a scaleShape implementation of the darkMatterProfile component from array.

<void> destroy() Finalize a scaleShape implementation of the darkMatterProfile component.

<void> dumpASCII() Dump the content of a darkMatterProfile component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a scaleShape member of the darkMatterProfile component.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a scaleShape implementation of the darkMatterProfile component class.

<logical> nullIsActive() Return true if the null implementation of the darkMatterProfile component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a darkMatterProfile component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a scaleShape implementation of the darkMatterProfile component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a scaleShape implementation of the darkMatterProfile component.

<void> postOutput(<double precision> time→) Perform post-output processing for a scaleShape implementation of the darkMatterProfile component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<double precision> scale() Get the value of the scale property of the scale implementation of the darkMatterProfile component using a deferred function.

<type(varying_string), allocatable, dimension(:) => matches> scaleAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the darkMatterProfile class that have the desired attributes for the scale property

<integer> scaleCount() Return a count of the number of scalar properties in the scale property of an scale implementation of the darkMatterProfile component class.

<void> scaleFunction() Set the function to be used for the get method of the scale property of the DarkMatterProfileScale component.

<double precision> scaleGrowthRate() Get the scaleGrowthRate property of an scale implementation of the darkMatterProfile component class.

<type(varying_string), allocatable, dimension(:) => matches> scaleGrowthRateAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the darkMatterProfile class that have the desired attributes for the scaleGrowthRate property

<logical> scaleGrowthRateIsGettable() Returns true if the scaleGrowthRate property is gettable for the darkMatterProfile component class.

<logical> scaleGrowthRateIsSettable() Specify whether the scaleGrowthRate property of the darkMatterProfile component is settable.

<double precision> scaleGrowthRateRateGet() Returns a zero rate for the scaleGrowthRate property for the darkMatterProfile component class.

<void> scaleGrowthRateSet(<double precision> setValue→) Set the scaleGrowthRate property of an scale implementation of the darkMatterProfile component class.

<void> scaleInactive() Indicate that the scale property of an scale implementation of the darkMatterProfile component class is inactive for differential equation solving.

<logical> scaleIsActive() Return true if the scale implementation of the darkMatterProfile component is the active choice.

<logical> scaleIsAttached() Return true if the deferred function used to get the scale property of the DarkMatterProfileScale component class has been attached.

<logical> scaleIsGettable() Returns true if the scale property is gettable for the darkMatterProfile component class.

<logical> scaleIsLimited() Get the scaleIsLimited property of an scale implementation of the darkMatterProfile component class.

<type(varying_string), allocatable, dimension(:) => matches> scaleIsLimitedAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the darkMatterProfile class that have the desired attributes for the scaleIsLimited property

<logical> scaleIsLimitedIsGettable() Returns true if the scaleIsLimited property is gettable for the darkMatterProfile component class.

<logical> scaleIsLimitedIsSettable() Specify whether the scaleIsLimited property of the darkMatterProfile component is settable.

<logical> scaleIsLimitedRateGet() Returns a zero rate for the scaleIsLimited property for the darkMatterProfile component class.

<void> scaleIsLimitedSet(<logical> setValue→) Set the scaleIsLimited property of an scale implementation of the darkMatterProfile component class.

<logical> scaleIsSettable() Specify whether the scale property of the darkMatterProfile component is settable.

<logical> scalePresetIsActive() Return true if the scalePreset implementation of the darkMatterProfile component is the active choice.

<void> scaleRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the scale property of an scale implementation of the darkMatterProfile component class.

<double precision> scaleRateGet() Get the rate of change of the scale property of an scale implementation of the darkMatterProfile component class.

<void> scaleScale(<double precision> setValue→) Set the absolute scale of the scale property of an scale implementation of the darkMatterProfile component class.

<void> scaleSet(<double precision> setValue→) Set the scale property of an scale implementation of the darkMatterProfile component class.

<logical> scaleShapeIsActive() Return true if the scaleShape implementation of the darkMatterProfile component is the active choice.

<double precision> scaleValue() Get the scale property of an scale implementation of the darkMatterProfile component class.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a scaleShape implementation of the darkMatterProfile component.

<void> serializeASCII() Serialize the contents of a scaleShape implementation of the darkMatterProfile component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a scaleShape implementation of the darkMatterProfile component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a scaleShape implementation of the darkMatterProfile component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a scaleShape implementation of the darkMatterProfile component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a scaleShape implementation of the darkMatterProfile component to XML.

<double precision> shape() Get the value of the shape property of the scaleShape implementation of the darkMatterProfile component using a deferred function.

<type(varying_string), allocatable, dimension(:) => matches> shapeAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the darkMatterProfile class that have the desired attributes for the shape property

<integer> shapeCount() Return a count of the number of scalar properties in the shape property of an scaleShape implementation of the darkMatterProfile component class.

<void> shapeFunction() Set the function to be used for the get method of the shape property of the DarkMatterProfileScaleShape component.

<double precision> shapeGrowthRate() Get the shapeGrowthRate property of an scaleShape implementation of the darkMatterProfile component class.

<type(varying_string), allocatable, dimension(:) => matches> shapeGrowthRateAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the darkMatterProfile class that have the desired attributes for the shapeGrowthRate property

<logical> shapeGrowthRateIsGettable() Returns true if the shapeGrowthRate property is gettable for the darkMatterProfile component class.

<logical> shapeGrowthRateIsSettable() Specify whether the shapeGrowthRate property of the darkMatterProfile component is settable.

<double precision> shapeGrowthRateRateGet() Returns a zero rate for the shapeGrowthRate property for the darkMatterProfile component class.

<void> shapeGrowthRateSet(<double precision> setValue→) Set the shapeGrowthRate property of an scaleShape implementation of the darkMatterProfile component class.

<void> shapeInactive() Indicate that the shape property of an scaleShape implementation of the darkMatterProfile component class is inactive for differential equation solving.

<logical> shapeIsAttached() Return true if the deferred function used to get the shape property of the DarkMatterProfileScaleShape component class has been attached.

<logical> shapeIsGettable() Returns true if the shape property is gettable for the darkMatterProfile component class.

<logical> shapeIsSettable() Specify whether the shape property of the darkMatterProfile component is settable.

<void> shapeRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the shape property of an scaleShape implementation of the darkMatterProfile component class.

<double precision> shapeRateGet() Get the rate of change of the shape property of an scaleShape implementation of the darkMatterProfile component class.

<void> shapeScale(<double precision> setValue→) Set the absolute scale of the shape property of an scaleShape implementation of the darkMatterProfile component class.

<void> shapeSet(<double precision> setValue→) Set the shape property of an scaleShape implementation of the darkMatterProfile component class.

<double precision> shapeValue() Get the shape property of an scaleShape implementation of the darkMatterProfile component class.

<integer(c_size_t)> sizeOf() Return the size in bytes of a scaleShape implementation of the darkMatterProfile component.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<type(varying_string)> type() Returns the type name for the scaleShape implementation of the darkMatterProfile component class.

nodeComponentDisk

<type(abundances)> abundancesGas() Returns the default value for the abundancesGas property for the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> abundancesGasAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the abundancesGas property

<integer> abundancesGasCount() Compute the count of evolvable quantities in the abundancesGas property of the DiskStandard component.

<logical> abundancesGasIsGettable() Returns true if the abundancesGas property is gettable for the disk component class.

<logical> abundancesGasIsSettable() Specify whether the abundancesGas property of the disk component is settable.

<void> abundancesGasRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptT [interruptProcedure]↔) Accept a rate set for the abundancesGas property of the disk component class. Trigger an interrupt to create the component.

<type(abundances)> abundancesGasRateGet() Returns a zero rate for the abundancesGas property for the disk component class.

<void> abundancesGasScale(<type(abundances)> value) Set the scale of the abundancesGas property of the DiskStandard component.

<void> abundancesGasSet(<type(abundances)> value) Set the abundancesGas property of the disk component.

<type(abundances)> abundancesStellar() Returns the default value for the abundancesStellar property for the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> abundancesStellarAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the abundancesStellar property

<integer> abundancesStellarCount() Compute the count of evolvable quantities in the abundancesStellar property of the DiskStandard component.

<logical> abundancesStellarIsGettable() Returns true if the abundancesStellar property is gettable for the disk component class.

<logical> abundancesStellarIsSettable() Specify whether the abundancesStellar property of the disk component is settable.

<void> abundancesStellarRate(<type(abundances)> value) Cumulate to the rate of the abundancesStellar property of the DiskStandard component.

<type(abundances)> abundancesStellarRateGet() Returns a zero rate for the abundancesStellar property for the disk component class.

<void> abundancesStellarScale(<type(abundances)> value) Set the scale of the abundancesStellar property of the DiskStandard component.

<void> abundancesStellarSet(<type(abundances)> value) Set the abundancesStellar property of the disk component.

<double precision> angularMomentum() Returns the default value for the angularMomentum property for the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> angularMomentumAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the angularMomentum property

<integer> angularMomentumCount() Compute the count of evolvable quantities in the angularMomentum property of the DiskStandard component.

<logical> angularMomentumIsGettable() Returns true if the angularMomentum property is gettable for the disk component class.

<logical> angularMomentumIsSettable() Specify whether the angularMomentum property of the disk component is settable.

<void> angularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interrup [interruptProcedure]↔) Accept a rate set for the angularMomentum property of the disk component class. Trigger an interrupt to create the component.

<double precision> angularMomentumRateGet() Returns a zero rate for the angularMomentum property for the disk component class.

<void> angularMomentumScale(<double precision> value) Set the scale of the angularMomentum property of the DiskStandard component.

<void> angularMomentumSet(<double precision> value) Set the angularMomentum property of the disk component.

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a generic disk component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Read properties from raw file.

<void> deserializeValues(<double(:)> array→, <integer> propertyType→) Deserialize the evolvable quantities from an array.

<void> destroy() Finalize a generic disk component.

<void> dumpASCII() Dump the content of a disk component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<double precision> fractionMassRetained() Returns the default value for the fractionMassRetained property for the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> fractionMassRetainedAttributeMatch(<logic [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the fractionMassRetained property

<integer> fractionMassRetainedCount() Compute the count of evolvable quantities in the fractionMassRetained property of the DiskStandard component.

<logical> fractionMassRetainedIsGettable() Returns true if the fractionMassRetained property is gettable for the disk component class.

<logical> fractionMassRetainedIsSettable() Specify whether the fractionMassRetained property of the disk component is settable.

<void> fractionMassRetainedRate(<double precision> value) Cumulate to the rate of the fractionMassRetained property of the DiskStandard component.

<double precision> fractionMassRetainedRateGet() Returns a zero rate for the fractionMassRetained property for the disk component class.

<void> fractionMassRetainedScale(<double precision> value) Set the scale of the fractionMassRetained property of the DiskStandard component.

<void> fractionMassRetainedSet(<double precision> value) Set the fractionMassRetained property of the disk component.

<double precision> halfMassRadius() Returns the default value for the halfMassRadius property for the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> halfMassRadiusAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the halfMassRadius property

<logical> halfMassRadiusIsGettable() Returns true if the halfMassRadius property is gettable for the disk component class.

<logical> halfMassRadiusIsSettable() Specify whether the halfMassRadius property of the disk component is settable.

<double precision> halfMassRadiusRateGet() Returns a zero rate for the halfMassRadius property for the disk component class.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a generic disk component.

<logical> isInitialized() Returns the default value for the isInitialized property for the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> isInitializedAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the isInitialized property

<logical> isInitializedIsGettable() Returns true if the isInitialized property is gettable for the disk component class.

<logical> isInitializedIsSettable() Specify whether the isInitialized property of the disk component is settable.

<logical> isInitializedRateGet() Returns a zero rate for the isInitialized property for the disk component class.

<void> isInitializedSet(<logical> value) Set the isInitialized property of the disk component.

<type(stellarLuminosities)> luminositiesStellar() Returns the default value for the luminositiesStellar property for the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> luminositiesStellarAttributeMatch(<logica
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the disk class that have the desired attributes for the luminositiesStellar property

<integer> luminositiesStellarCount() Compute the count of evolvable quantities in the luminositiesStellar property of the DiskStandard component.

<logical> luminositiesStellarIsGettable() Returns true if the luminositiesStellar property is gettable for the disk component class.

<logical> luminositiesStellarIsSettable() Specify whether the luminositiesStellar property of the disk component is settable.

<void> luminositiesStellarRate(<type(stellarLuminosities)> value) Cumulate to the rate of the luminositiesStellar property of the DiskStandard component.

<type(stellarLuminosities)> luminositiesStellarRateGet() Returns a zero rate for the luminositiesStellar property for the disk component class.

<void> luminositiesStellarScale(<type(stellarLuminosities)> value) Set the scale of the luminositiesStellar property of the DiskStandard component.

<void> luminositiesStellarSet(<type(stellarLuminosities)> value) Set the luminositiesStellar property of the disk component.

<double precision> massGas() Returns the default value for the massGas property for the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> massGasAttributeMatch(<logical>
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the disk class that have the desired attributes for the massGas property

<integer> massGasCount() Compute the count of evolvable quantities in the massGas property of the DiskStandard component.

<logical> massGasIsGettable() Returns true if the massGas property is gettable for the disk component class.

<logical> massGasIsSettable() Specify whether the massGas property of the disk component is settable.

<void> massGasRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)>
[interruptProcedure]↔) Accept a rate set for the massGas property of the disk component class. Trigger an interrupt to create the component.

<double precision> massGasRateGet() Returns a zero rate for the massGas property for the disk component class.

<void> massGasScale(<double precision> value) Set the scale of the massGas property of the DiskStandard component.

<void> massGasSet(<double precision> value) Set the massGas property of the disk component.

<double precision> massStellar() Returns the default value for the massStellar property for the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> massStellarAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the massStellar property

<integer> massStellarCount() Compute the count of evolvable quantities in the massStellar property of the DiskStandard component.

<double precision> massStellarFormed() Returns the default value for the massStellarFormed property for the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> massStellarFormedAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the massStellarFormed property

<integer> massStellarFormedCount() Compute the count of evolvable quantities in the massStellarFormed property of the DiskStandard component.

<logical> massStellarFormedIsGettable() Returns true if the massStellarFormed property is gettable for the disk component class.

<logical> massStellarFormedIsSettable() Specify whether the massStellarFormed property of the disk component is settable.

<void> massStellarFormedRate(<double precision> value) Cumulate to the rate of the massStellarFormed property of the DiskStandard component.

<double precision> massStellarFormedRateGet() Returns a zero rate for the massStellarFormed property for the disk component class.

<void> massStellarFormedScale(<double precision> value) Set the scale of the massStellarFormed property of the DiskStandard component.

<void> massStellarFormedSet(<double precision> value) Set the massStellarFormed property of the disk component.

<logical> massStellarIsGettable() Returns true if the massStellar property is gettable for the disk component class.

<logical> massStellarIsSettable() Specify whether the massStellar property of the disk component is settable.

<void> massStellarRate(<double precision> value) Cumulate to the rate of the massStellar property of the DiskStandard component.

<double precision> massStellarRateGet() Returns a zero rate for the massStellar property for the disk component class.

<void> massStellarScale(<double precision> value) Set the scale of the massStellar property of the DiskStandard component.

<void> massStellarSet(<double precision> value) Set the massStellar property of the disk component.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a nodeComponent object.

<logical> nullIsActive() Return true if the null implementation of the disk component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a disk component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a generic disk component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Establish the names of properties to output for a generic disk component.

<void> postOutput(<double precision> time→) Perform post-output processing of a disk component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<double precision> radius() Returns the default value for the radius property for the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> radiusAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the radius property

<logical> radiusIsGettable() Returns true if the radius property is gettable for the disk component class.

<logical> radiusIsSettable() Specify whether the radius property of the disk component is settable.

<double precision> radiusRateGet() Returns a zero rate for the radius property for the disk component class.

<void> radiusSet(<double precision> value) Set the radius property of the disk component.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets() Set offsets into serialization arrays.

<void> serializeASCII() Serialize the content of a disk component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the number of evolvable quantities to be evolved.

<void> serializeRaw(<integer> fileHandle→) Generate a binary dump of all properties.

<void> serializeValues(<double(:)> array←, <integer> propertyType→) Serialize the evolvable quantities to an array.

<void> serializeXML() Generate an XML dump of all properties.

<integer(c_size_t)> sizeOf() Return the size in bytes of a nodeComponentDisk component.

<logical> standardIsActive() Return true if the standard implementation of the disk component is the active choice.

<type(history)> starFormationHistory() Returns the default value for the starFormationHistory property for the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> starFormationHistoryAttributeMatch(<logic [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the starFormationHistory property

<integer> starFormationHistoryCount() Compute the count of evolvable quantities in the starFormationHistory property of the DiskStandard component.

<logical> starFormationHistoryIsGettable() Returns true if the starFormationHistory property is gettable for the disk component class.

<logical> starFormationHistoryIsSettable() Specify whether the starFormationHistory property of the disk component is settable.

<void> starFormationHistoryRate(<type(history)> value) Cumulate to the rate of the starFormationHistory property of the DiskStandard component.

<type(history)> starFormationHistoryRateGet() Returns a zero rate for the starFormationHistory property for the disk component class.

<void> starFormationHistoryScale(<type(history)> value) Set the scale of the starFormationHistory property of the DiskStandard component.

<void> starFormationHistorySet(<type(history)> value) Set the starFormationHistory property of the disk component.

<double precision> starFormationRate() Returns the default value for the starFormationRate property for the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> starFormationRateAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the starFormationRate property

<logical> starFormationRateIsGettable() Returns true if the starFormationRate property is gettable for the disk component class.

<logical> starFormationRateIsSettable() Specify whether the starFormationRate property of the disk component is settable.

<double precision> starFormationRateRateGet() Returns a zero rate for the starFormationRate property for the disk component class.

<type(history)> stellarPropertiesHistory() Returns the default value for the stellarPropertiesHistory property for the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> stellarPropertiesHistoryAttributeMatch(<l [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the stellarPropertiesHistory property

<integer> stellarPropertiesHistoryCount() Compute the count of evolvable quantities in the stellarPropertiesHist property of the DiskStandard component.

<logical> stellarPropertiesHistoryIsGettable() Returns true if the stellarPropertiesHistory property is gettable for the disk component class.

<logical> stellarPropertiesHistoryIsSettable() Specify whether the stellarPropertiesHistory property of the disk component is settable.

<void> stellarPropertiesHistoryRate(<type(history)> value) Cumulate to the rate of the stellarPropertiesHisto property of the DiskStandard component.

<type(history)> stellarPropertiesHistoryRateGet() Returns a zero rate for the stellarPropertiesHistory property for the disk component class.

<void> stellarPropertiesHistoryScale(<type(history)> value) Set the scale of the stellarPropertiesHistory property of the DiskStandard component.

<void> stellarPropertiesHistorySet(<type(history)> value) Set the stellarPropertiesHistory property of the disk component.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<type(varying_string)> type() Returns the type name for the disk component class.

<double precision> velocity() Returns the default value for the velocity property for the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> velocityAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the velocity property

<logical> velocityIsGettable() Returns true if the velocity property is gettable for the disk component class.

<logical> velocityIsSettable() Specify whether the velocity property of the disk component is settable.

<double precision> velocityRateGet() Returns a zero rate for the velocity property for the disk component class.

<void> velocitySet(<double precision> value) Set the velocity property of the disk component.

<logical> verySimpleIsActive() Return true if the verySimple implementation of the disk component is the active choice.

<logical> verySimpleSizeIsActive() Return true if the verySimpleSize implementation of the disk component is the active choice.

nodeComponentDiskNull

<type(abundances)> abundancesGas() Returns the default value for the abundancesGas property for the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> abundancesGasAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the abundancesGas property

<integer> abundancesGasCount() Compute the count of evolvable quantities in the abundancesGas property of the DiskStandard component.

<logical> abundancesGasIsGettable() Returns true if the abundancesGas property is gettable for the disk component class.

<logical> abundancesGasIsSettable() Specify whether the abundancesGas property of the disk component is settable.

<void> abundancesGasRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptT [interruptProcedure]↔) Accept a rate set for the abundancesGas property of the disk component class. Trigger an interrupt to create the component.

<type(abundances)> abundancesGasRateGet() Returns a zero rate for the abundancesGas property for the disk component class.

<void> abundancesGasScale(<type(abundances)> value) Set the scale of the abundancesGas property of the DiskStandard component.

<void> abundancesGasSet(<type(abundances)> value) Set the abundancesGas property of the disk component.

<type(abundances)> abundancesStellar() Returns the default value for the abundancesStellar property for the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> abundancesStellarAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the abundancesStellar property

`<integer>` `abundancesStellarCount()` Compute the count of evolvable quantities in the `abundancesStellar` property of the `DiskStandard` component.

`<logical>` `abundancesStellarIsGettable()` Returns true if the `abundancesStellar` property is gettable for the `disk` component class.

`<logical>` `abundancesStellarIsSettable()` Specify whether the `abundancesStellar` property of the `disk` component is settable.

`<void>` `abundancesStellarRate(<type(abundances)> value)` Cumulate to the rate of the `abundancesStellar` property of the `DiskStandard` component.

`<type(abundances)>` `abundancesStellarRateGet()` Returns a zero rate for the `abundancesStellar` property for the `disk` component class.

`<void>` `abundancesStellarScale(<type(abundances)> value)` Set the scale of the `abundancesStellar` property of the `DiskStandard` component.

`<void>` `abundancesStellarSet(<type(abundances)> value)` Set the `abundancesStellar` property of the `disk` component.

`<double precision>` `angularMomentum()` Returns the default value for the `angularMomentum` property for the `disk` component class.

`<type(varying_string), allocatable, dimension(:) => matches>` `angularMomentumAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)` Return a text list of component implementations in the `disk` class that have the desired attributes for the `angularMomentum` property

`<integer>` `angularMomentumCount()` Compute the count of evolvable quantities in the `angularMomentum` property of the `DiskStandard` component.

`<logical>` `angularMomentumIsGettable()` Returns true if the `angularMomentum` property is gettable for the `disk` component class.

`<logical>` `angularMomentumIsSettable()` Specify whether the `angularMomentum` property of the `disk` component is settable.

`<void>` `angularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interrup [interruptProcedure]↔)` Accept a rate set for the `angularMomentum` property of the `disk` component class. Trigger an interrupt to create the component.

`<double precision>` `angularMomentumRateGet()` Returns a zero rate for the `angularMomentum` property for the `disk` component class.

`<void>` `angularMomentumScale(<double precision> value)` Set the scale of the `angularMomentum` property of the `DiskStandard` component.

`<void>` `angularMomentumSet(<double precision> value)` Set the `angularMomentum` property of the `disk` component.

`<void>` `assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→)` Assign a node component to another node component.

`<void>` `builder(<*type(node)> componentDefinition→)` Build a `null` implementation of the `disk` component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a null implementation of the disk component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a null implementation of the disk component from array.

<void> destroy() Finalize a null implementation of the disk component.

<void> dumpASCII() Dump the content of a disk component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<double precision> fractionMassRetained() Returns the default value for the fractionMassRetained property for the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> fractionMassRetainedAttributeMatch(<logic [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the fractionMassRetained property

<integer> fractionMassRetainedCount() Compute the count of evolvable quantities in the fractionMassRetained property of the DiskStandard component.

<logical> fractionMassRetainedIsGettable() Returns true if the fractionMassRetained property is gettable for the disk component class.

<logical> fractionMassRetainedIsSettable() Specify whether the fractionMassRetained property of the disk component is settable.

<void> fractionMassRetainedRate(<double precision> value) Cumulate to the rate of the fractionMassRetained property of the DiskStandard component.

<double precision> fractionMassRetainedRateGet() Returns a zero rate for the fractionMassRetained property for the disk component class.

<void> fractionMassRetainedScale(<double precision> value) Set the scale of the fractionMassRetained property of the DiskStandard component.

<void> fractionMassRetainedSet(<double precision> value) Set the fractionMassRetained property of the disk component.

<double precision> halfMassRadius() Returns the default value for the halfMassRadius property for the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> halfMassRadiusAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the halfMassRadius property

<logical> halfMassRadiusIsGettable() Returns true if the halfMassRadius property is gettable for the disk component class.

<logical> halfMassRadiusIsSettable() Specify whether the halfMassRadius property of the disk component is settable.

<double precision> halfMassRadiusRateGet() Returns a zero rate for the halfMassRadius property for the disk component class.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a null member of the disk component.

<logical> isInitialized() Returns the default value for the isInitialized property for the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> isInitializedAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the isInitialized property

<logical> isInitializedIsGettable() Returns true if the isInitialized property is gettable for the disk component class.

<logical> isInitializedIsSettable() Specify whether the isInitialized property of the disk component is settable.

<logical> isInitializedRateGet() Returns a zero rate for the isInitialized property for the disk component class.

<void> isInitializedSet(<logical> value) Set the isInitialized property of the disk component.

<type(stellarLuminosities)> luminositiesStellar() Returns the default value for the luminositiesStellar property for the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> luminositiesStellarAttributeMatch(<logica [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the luminositiesStellar property

<integer> luminositiesStellarCount() Compute the count of evolvable quantities in the luminositiesStellar property of the DiskStandard component.

<logical> luminositiesStellarIsGettable() Returns true if the luminositiesStellar property is gettable for the disk component class.

<logical> luminositiesStellarIsSettable() Specify whether the luminositiesStellar property of the disk component is settable.

<void> luminositiesStellarRate(<type(stellarLuminosities)> value) Cumulate to the rate of the luminositiesStellar property of the DiskStandard component.

<type(stellarLuminosities)> luminositiesStellarRateGet() Returns a zero rate for the luminositiesStellar property for the disk component class.

<void> luminositiesStellarScale(<type(stellarLuminosities)> value) Set the scale of the luminositiesStellar property of the DiskStandard component.

<void> luminositiesStellarSet(<type(stellarLuminosities)> value) Set the luminositiesStellar property of the disk component.

<double precision> massGas() Returns the default value for the massGas property for the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> massGasAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the massGas property

<integer> massGasCount() Compute the count of evolvable quantities in the massGas property of the DiskStandard component.

<logical> massGasIsGettable() Returns true if the massGas property is gettable for the disk component class.

<logical> massGasIsSettable() Specify whether the massGas property of the disk component is settable.

<void> massGasRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accept a rate set for the massGas property of the disk component class. Trigger an interrupt to create the component.

<double precision> massGasRateGet() Returns a zero rate for the massGas property for the disk component class.

<void> massGasScale(<double precision> value) Set the scale of the massGas property of the DiskStandard component.

<void> massGasSet(<double precision> value) Set the massGas property of the disk component.

<double precision> massStellar() Returns the default value for the massStellar property for the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> massStellarAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the massStellar property

`<integer> massStellarCount()` Compute the count of evolvable quantities in the `massStellar` property of the `DiskStandard` component.

`<double precision> massStellarFormed()` Returns the default value for the `massStellarFormed` property for the `disk` component class.

`<type(varying_string), allocatable, dimension(:) => matches> massStellarFormedAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)` Return a text list of component implementations in the `disk` class that have the desired attributes for the `massStellarFormed` property

`<integer> massStellarFormedCount()` Compute the count of evolvable quantities in the `massStellarFormed` property of the `DiskStandard` component.

`<logical> massStellarFormedIsGettable()` Returns true if the `massStellarFormed` property is gettable for the `disk` component class.

`<logical> massStellarFormedIsSettable()` Specify whether the `massStellarFormed` property of the `disk` component is settable.

`<void> massStellarFormedRate(<double precision> value)` Cumulate to the rate of the `massStellarFormed` property of the `DiskStandard` component.

`<double precision> massStellarFormedRateGet()` Returns a zero rate for the `massStellarFormed` property for the `disk` component class.

`<void> massStellarFormedScale(<double precision> value)` Set the scale of the `massStellarFormed` property of the `DiskStandard` component.

`<void> massStellarFormedSet(<double precision> value)` Set the `massStellarFormed` property of the `disk` component.

`<logical> massStellarIsGettable()` Returns true if the `massStellar` property is gettable for the `disk` component class.

`<logical> massStellarIsSettable()` Specify whether the `massStellar` property of the `disk` component is settable.

`<void> massStellarRate(<double precision> value)` Cumulate to the rate of the `massStellar` property of the `DiskStandard` component.

`<double precision> massStellarRateGet()` Returns a zero rate for the `massStellar` property for the `disk` component class.

`<void> massStellarScale(<double precision> value)` Set the scale of the `massStellar` property of the `DiskStandard` component.

`<void> massStellarSet(<double precision> value)` Set the `massStellar` property of the `disk` component.

`<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→)` Return the name of the property of given index for a `null` implementation of the `disk` component class.

`<logical> nullIsActive()` Return true if the null implementation of the disk component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a disk component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a null implementation of the disk component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a null implementation of the disk component.

<void> postOutput(<double precision> time→) Perform post-output processing of a disk component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<double precision> radius() Returns the default value for the radius property for the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> radiusAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the radius property

<logical> radiusIsGettable() Returns true if the radius property is gettable for the disk component class.

<logical> radiusIsSettable() Specify whether the radius property of the disk component is settable.

<double precision> radiusRateGet() Returns a zero rate for the radius property for the disk component class.

<void> radiusSet(<double precision> value) Set the radius property of the disk component.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a null implementation of the disk component.

<void> serializeASCII() Serialize the contents of a null implementation of the disk component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a null implementation of the disk component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a null implementation of the disk component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a null implementation of the disk component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a null implementation of the disk component to XML.

<integer(c_size_t)> sizeOf() Return the size in bytes of a null implementation of the disk component.

<logical> standardIsActive() Return true if the standard implementation of the disk component is the active choice.

<type(history)> starFormationHistory() Returns the default value for the starFormationHistory property for the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> starFormationHistoryAttributeMatch(<logic
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the disk class that have the desired attributes for the starFormationHistory property

<integer> starFormationHistoryCount() Compute the count of evolvable quantities in the starFormationHistory property of the DiskStandard component.

<logical> starFormationHistoryIsGettable() Returns true if the starFormationHistory property is gettable for the disk component class.

<logical> starFormationHistoryIsSettable() Specify whether the starFormationHistory property of the disk component is settable.

<void> starFormationHistoryRate(<type(history)> value) Cumulate to the rate of the starFormationHistory property of the DiskStandard component.

<type(history)> starFormationHistoryRateGet() Returns a zero rate for the starFormationHistory property for the disk component class.

<void> starFormationHistoryScale(<type(history)> value) Set the scale of the starFormationHistory property of the DiskStandard component.

<void> starFormationHistorySet(<type(history)> value) Set the starFormationHistory property of the disk component.

<double precision> starFormationRate() Returns the default value for the starFormationRate property for the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> starFormationRateAttributeMatch(<logical>
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the disk class that have the desired attributes for the starFormationRate property

<logical> starFormationRateIsGettable() Returns true if the starFormationRate property is gettable for the disk component class.

<logical> starFormationRateIsSettable() Specify whether the starFormationRate property of the disk component is settable.

<double precision> starFormationRateRateGet() Returns a zero rate for the starFormationRate property for the disk component class.

<type(history)> stellarPropertiesHistory() Returns the default value for the stellarPropertiesHistory property for the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> stellarPropertiesHistoryAttributeMatch(<l [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the stellarPropertiesHistory property

<integer> stellarPropertiesHistoryCount() Compute the count of evolvable quantities in the stellarPropertiesHist property of the DiskStandard component.

<logical> stellarPropertiesHistoryIsGettable() Returns true if the stellarPropertiesHistory property is gettable for the disk component class.

<logical> stellarPropertiesHistoryIsSettable() Specify whether the stellarPropertiesHistory property of the disk component is settable.

<void> stellarPropertiesHistoryRate(<type(history)> value) Cumulate to the rate of the stellarPropertiesHisto property of the DiskStandard component.

<type(history)> stellarPropertiesHistoryRateGet() Returns a zero rate for the stellarPropertiesHistory property for the disk component class.

<void> stellarPropertiesHistoryScale(<type(history)> value) Set the scale of the stellarPropertiesHistory property of the DiskStandard component.

<void> stellarPropertiesHistorySet(<type(history)> value) Set the stellarPropertiesHistory property of the disk component.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<type(varying_string)> type() Returns the type name for the null implementation of the disk component class.

<double precision> velocity() Returns the default value for the velocity property for the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> velocityAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the velocity property

<logical> velocityIsGettable() Returns true if the velocity property is gettable for the disk component class.

<logical> velocityIsSettable() Specify whether the velocity property of the disk component is settable.

<double precision> velocityRateGet() Returns a zero rate for the velocity property for the disk component class.

<void> velocitySet(<double precision> value) Set the velocity property of the disk component.

<logical> verySimpleIsActive() Return true if the verySimple implementation of the disk component is the active choice.

<logical> verySimpleSizeIsActive() Return true if the verySimpleSize implementation of the disk component is the active choice.

nodeComponentDiskStandard

<type(abundances)> abundancesGas() Get the abundancesGas property of an standard implementation of the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> abundancesGasAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the abundancesGas property

<integer> abundancesGasCount() Return a count of the number of scalar properties in the abundancesGas property of an standard implementation of the disk component class.

<void> abundancesGasInactive() Indicate that the abundancesGas property of an standard implementation of the disk component class is inactive for differential equation solving.

<logical> abundancesGasIsGettable() Returns true if the abundancesGas property is gettable for the disk component class.

<logical> abundancesGasIsSettable() Specify whether the abundancesGas property of the disk component is settable.

<void> abundancesGasRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptT [interruptProcedure]↔) Accumulate to the rate of change of the abundancesGas property of an standard implementation of the disk component class.

<type(abundances)> abundancesGasRateGet() Get the rate of change of the abundancesGas property of an standard implementation of the disk component class.

<void> abundancesGasScale(<type(abundances)> setValue→) Set the absolute scale of the abundancesGas property of an standard implementation of the disk component class.

<void> abundancesGasSet(<type(abundances)> setValue→) Set the abundancesGas property of an standard implementation of the disk component class.

<type(abundances)> abundancesStellar() Get the abundancesStellar property of an standard implementation of the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> abundancesStellarAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the abundancesStellar property

<integer> abundancesStellarCount() Return a count of the number of scalar properties in the abundancesStellar property of an standard implementation of the disk component class.

<void> abundancesStellarInactive() Indicate that the abundancesStellar property of an standard implementation of the disk component class is inactive for differential equation solving.

<logical> abundancesStellarIsGettable() Returns true if the abundancesStellar property is gettable for the disk component class.

<logical> abundancesStellarIsSettable() Specify whether the abundancesStellar property of the disk component is settable.

<void> abundancesStellarRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interr [interruptProcedure]↔) Accumulate to the rate of change of the abundancesStellar property of an standard implementation of the disk component class.

<type(abundances)> abundancesStellarRateGet() Get the rate of change of the abundancesStellar property of an standard implementation of the disk component class.

<void> abundancesStellarScale(<type(abundances)> setValue→) Set the absolute scale of the abundancesStellar property of an standard implementation of the disk component class.

<void> abundancesStellarSet(<type(abundances)> setValue→) Set the abundancesStellar property of an standard implementation of the disk component class.

<double precision> angularMomentum() Get the angularMomentum property of an standard implementation of the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> angularMomentumAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the angularMomentum property

<integer> angularMomentumCount() Return a count of the number of scalar properties in the angularMomentum property of an standard implementation of the disk component class.

<void> angularMomentumInactive() Indicate that the angularMomentum property of an standard implementation of the disk component class is inactive for differential equation solving.

<logical> angularMomentumIsGettable() Returns true if the angularMomentum property is gettable for the disk component class.

<logical> angularMomentumIsSettable() Specify whether the angularMomentum property of the disk component is settable.

<void> angularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interrup [interruptProcedure]↔) Accumulate to the rate of change of the angularMomentum property of an standard implementation of the disk component class.

<double precision> angularMomentumRateGet() Get the rate of change of the angularMomentum property of an standard implementation of the disk component class.

<void> angularMomentumScale(<double precision> setValue→) Set the absolute scale of the angularMomentum property of an standard implementation of the disk component class.

<void> angularMomentumSet(<double precision> setValue→) Set the angularMomentum property of an standard implementation of the disk component class.

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> attachPipes() Attach pipes to the standard disk component.

<void> builder(<*type(node)> componentDefinition→) Build a standard implementation of the disk component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a standard implementation of the disk component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a standard implementation of the disk component from array.

<void> destroy() Finalize a standard implementation of the disk component.

<void> dumpASCII() Dump the content of a disk component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<double precision> fractionMassRetained() Get the fractionMassRetained property of an standard implementation of the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> fractionMassRetainedAttributeMatch(<logic [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the fractionMassRetained property

<integer> fractionMassRetainedCount() Return a count of the number of scalar properties in the fractionMassRetained property of an standard implementation of the disk component class.

<void> fractionMassRetainedInactive() Indicate that the fractionMassRetained property of an standard implementation of the disk component class is inactive for differential equation solving.

<logical> fractionMassRetainedIsGettable() Returns true if the fractionMassRetained property is gettable for the disk component class.

<logical> fractionMassRetainedIsSettable() Specify whether the fractionMassRetained property of the disk component is settable.

<void> fractionMassRetainedRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the fractionMassRetained property of an standard implementation of the disk component class.

<double precision> fractionMassRetainedRateGet() Get the rate of change of the fractionMassRetained property of an standard implementation of the disk component class.

<void> fractionMassRetainedScale(<double precision> setValue→) Set the absolute scale of the fractionMassRetained property of an standard implementation of the disk component class.

<void> fractionMassRetainedSet(<double precision> setValue→) Set the fractionMassRetained property of an standard implementation of the disk component class.

<double precision> halfMassRadius() Returns the default value for the halfMassRadius property for the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> halfMassRadiusAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the halfMassRadius property

<logical> halfMassRadiusIsGettable() Returns true if the halfMassRadius property is gettable for the disk component class.

<logical> halfMassRadiusIsSettable() Specify whether the halfMassRadius property of the disk component is settable.

<double precision> halfMassRadiusRateGet() Returns a zero rate for the halfMassRadius property for the disk component class.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a standard member of the disk component.

<logical> isInitialized() Get the isInitialized property of an standard implementation of the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> isInitializedAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the isInitialized property

<logical> isInitializedIsGettable() Returns true if the isInitialized property is gettable for the disk component class.

<logical> isInitializedIsSettable() Specify whether the isInitialized property of the disk component is settable.

<logical> isInitializedRateGet() Returns a zero rate for the isInitialized property for the disk component class.

<void> isInitializedSet(<logical> setValue→) Set the isInitialized property of an standard implementation of the disk component class.

<type(stellarLuminosities)> luminositiesStellar() Get the luminositiesStellar property of an standard implementation of the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> luminositiesStellarAttributeMatch(<logica [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the luminositiesStellar property

<integer> luminositiesStellarCount() Return a count of the number of scalar properties in the luminositiesStellar property of an standard implementation of the disk component class.

<void> luminositiesStellarInactive() Indicate that the luminositiesStellar property of an standard implementation of the disk component class is inactive for differential equation solving.

<logical> luminositiesStellarIsGettable() Returns true if the luminositiesStellar property is gettable for the disk component class.

<logical> luminositiesStellarIsSettable() Specify whether the luminositiesStellar property of the disk component is settable.

<void> luminositiesStellarRate(<type(stellarLuminosities)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the luminositiesStellar property of an standard implementation of the disk component class.

<type(stellarLuminosities)> luminositiesStellarRateGet() Get the rate of change of the luminositiesStellar property of an standard implementation of the disk component class.

<void> luminositiesStellarScale(<type(stellarLuminosities)> setValue→) Set the absolute scale of the luminositiesStellar property of an standard implementation of the disk component class.

<void> luminositiesStellarSet(<type(stellarLuminosities)> setValue→) Set the luminositiesStellar property of an standard implementation of the disk component class.

<double precision> massGas() Get the massGas property of an standard implementation of the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> massGasAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the massGas property

<integer> massGasCount() Return a count of the number of scalar properties in the massGas property of an standard implementation of the disk component class.

<void> massGasInactive() Indicate that the massGas property of an standard implementation of the disk component class is inactive for differential equation solving.

<logical> massGasIsGettable() Returns true if the massGas property is gettable for the disk component class.

<logical> massGasIsSettable() Specify whether the massGas property of the disk component is settable.

<void> massGasRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the massGas property of an standard implementation of the disk component class.

<double precision> massGasRateGet() Get the rate of change of the massGas property of an standard implementation of the disk component class.

<void> massGasScale(<double precision> setValue→) Set the absolute scale of the massGas property of an standard implementation of the disk component class.

<void> massGasSet(<double precision> setValue→) Set the massGas property of an standard implementation of the disk component class.

<double precision> massStellar() Get the massStellar property of an standard implementation of the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> massStellarAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the massStellar property

<integer> massStellarCount() Return a count of the number of scalar properties in the massStellar property of an standard implementation of the disk component class.

<double precision> massStellarFormed() Get the massStellarFormed property of an standard implementation of the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> massStellarFormedAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the massStellarFormed property

<integer> massStellarFormedCount() Return a count of the number of scalar properties in the massStellarFormed property of an standard implementation of the disk component class.

<void> massStellarFormedInactive() Indicate that the massStellarFormed property of an standard implementation of the disk component class is inactive for differential equation solving.

<logical> massStellarFormedIsGettable() Returns true if the massStellarFormed property is gettable for the disk component class.

<logical> massStellarFormedIsSettable() Specify whether the massStellarFormed property of the disk component is settable.

<void> massStellarFormedRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interr [interruptProcedure]↔) Accumulate to the rate of change of the massStellarFormed property of an standard implementation of the disk component class.

<double precision> massStellarFormedRateGet() Get the rate of change of the massStellarFormed property of an standard implementation of the disk component class.

<void> massStellarFormedScale(<double precision> setValue→) Set the absolute scale of the massStellarFormed property of an standard implementation of the disk component class.

<void> massStellarFormedSet(<double precision> setValue→) Set the massStellarFormed property of an standard implementation of the disk component class.

<void> massStellarInactive() Indicate that the massStellar property of an standard implementation of the disk component class is inactive for differential equation solving.

<logical> massStellarIsGettable() Returns true if the massStellar property is gettable for the disk component class.

<logical> massStellarIsSettable() Specify whether the massStellar property of the disk component is settable.

<void> massStellarRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTas [interruptProcedure]↔) Accumulate to the rate of change of the massStellar property of an standard implementation of the disk component class.

<double precision> massStellarRateGet() Get the rate of change of the massStellar property of an standard implementation of the disk component class.

<void> massStellarScale(<double precision> setValue→) Set the absolute scale of the massStellar property of an standard implementation of the disk component class.

<void> massStellarSet(<double precision> setValue→) Set the massStellar property of an standard implementation of the disk component class.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a standard implementation of the disk component class.

<logical> nullIsActive() Return true if the null implementation of the disk component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a disk component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a standard implementation of the disk component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a standard implementation of the disk component.

<void> postOutput(<double precision> time→) Perform post-output processing for a standard implementation of the disk component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<double precision> radius() Get the radius property of an standard implementation of the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> radiusAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the radius property

<logical> radiusIsGettable() Returns true if the radius property is gettable for the disk component class.

<logical> radiusIsSettable() Specify whether the radius property of the disk component is settable.

<double precision> radiusRateGet() Returns a zero rate for the radius property for the disk component class.

<void> radiusSet(<double precision> setValue→) Set the radius property of an standard implementation of the disk component class.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a standard implementation of the disk component.

<void> serializeASCII() Serialize the contents of a standard implementation of the disk component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a standard implementation of the disk component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a standard implementation of the disk component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a standard implementation of the disk component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a standard implementation of the disk component to XML.

<integer(c_size_t)> sizeOf() Return the size in bytes of a standard implementation of the disk component.

<logical> standardIsActive() Return true if the standard implementation of the disk component is the active choice.

<type(history)> starFormationHistory() Get the starFormationHistory property of an standard implementation of the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> starFormationHistoryAttributeMatch(<logic [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the starFormationHistory property

<integer> starFormationHistoryCount() Return a count of the number of scalar properties in the starFormationHistory property of an standard implementation of the disk component class.

<void> starFormationHistoryInactive() Indicate that the starFormationHistory property of an standard implementation of the disk component class is inactive for differential equation solving.

<logical> starFormationHistoryIsGettable() Returns true if the starFormationHistory property is gettable for the disk component class.

<logical> starFormationHistoryIsSettable() Specify whether the starFormationHistory property of the disk component is settable.

<void> starFormationHistoryRate(<type(history)> setValue→, <logical> [interrupt]↔, <*procedure(interr [interruptProcedure]↔) Accumulate to the rate of change of the starFormationHistory property of an standard implementation of the disk component class.

<type(history)> starFormationHistoryRateGet() Get the rate of change of the starFormationHistory property of an standard implementation of the disk component class.

<void> starFormationHistoryScale(<type(history)> setValue→) Set the absolute scale of the starFormationHistory property of an standard implementation of the disk component class.

<void> starFormationHistorySet(<type(history)> setValue→) Set the starFormationHistory property of an standard implementation of the disk component class.

<double precision> starFormationRate() Get the value of the starFormationRate property of the standard implementation of the disk component using a deferred function.

<type(varying_string), allocatable, dimension(:) => matches> starFormationRateAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the starFormationRate property

<void> starFormationRateFunction() Set the function to be used for the get method of the starFormationRate property of the DiskStandard component.

<logical> starFormationRateIsAttached() Return true if the deferred function used to get the starFormationRate property of the DiskStandard component class has been attached.

<logical> starFormationRateIsGettable() Returns true if the starFormationRate property is gettable for the disk component class.

<logical> starFormationRateIsSettable() Specify whether the starFormationRate property of the disk component is settable.

<double precision> starFormationRateRateGet() Returns a zero rate for the starFormationRate property for the disk component class.

<type(history)> stellarPropertiesHistory() Get the stellarPropertiesHistory property of an standard implementation of the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> stellarPropertiesHistoryAttributeMatch(<l
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the disk class that have the desired attributes for the stellarPropertiesHistory property

<integer> stellarPropertiesHistoryCount() Return a count of the number of scalar properties in the stellarPropertiesHistory property of an standard implementation of the disk component class.

<void> stellarPropertiesHistoryInactive() Indicate that the stellarPropertiesHistory property of an standard implementation of the disk component class is inactive for differential equation solving.

<logical> stellarPropertiesHistoryIsGettable() Returns true if the stellarPropertiesHistory property is gettable for the disk component class.

<logical> stellarPropertiesHistoryIsSettable() Specify whether the stellarPropertiesHistory property of the disk component is settable.

<void> stellarPropertiesHistoryRate(<type(history)> setValue→, <logical> [interrupt]↔,
<*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the stellarPropertiesHistory property of an standard implementation of the disk component class.

<type(history)> stellarPropertiesHistoryRateGet() Get the rate of change of the stellarPropertiesHistory property of an standard implementation of the disk component class.

<void> stellarPropertiesHistoryScale(<type(history)> setValue→) Set the absolute scale of the stellarPropertiesHistory property of an standard implementation of the disk component class.

<void> stellarPropertiesHistorySet(<type(history)> setValue→) Set the stellarPropertiesHistory property of an standard implementation of the disk component class.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→,
<massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<type(varying_string)> type() Returns the type name for the standard implementation of the disk component class.

<double precision> velocity() Get the velocity property of an standard implementation of the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> velocityAttributeMatch(<logical>
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the disk class that have the desired attributes for the velocity property

<logical> velocityIsGettable() Returns true if the velocity property is gettable for the disk component class.

<logical> velocityIsSettable() Specify whether the velocity property of the disk component is settable.

<double precision> velocityRateGet() Returns a zero rate for the velocity property for the disk component class.

<void> velocitySet(<double precision> setValue→) Set the velocity property of an standard implementation of the disk component class.

<logical> verySimpleIsActive() Return true if the verySimple implementation of the disk component is the active choice.

<logical> verySimpleSizeIsActive() Return true if the verySimpleSize implementation of the disk component is the active choice.

nodeComponentDiskVerySimple

<type(abundances)> abundancesGas() Get the abundancesGas property of an verySimple implementation of the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> abundancesGasAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the abundancesGas property

<integer> abundancesGasCount() Return a count of the number of scalar properties in the abundancesGas property of an verySimple implementation of the disk component class.

<void> abundancesGasInactive() Indicate that the abundancesGas property of an verySimple implementation of the disk component class is inactive for differential equation solving.

<logical> abundancesGasIsGettable() Returns true if the abundancesGas property is gettable for the disk component class.

<logical> abundancesGasIsSettable() Specify whether the abundancesGas property of the disk component is settable.

<void> abundancesGasRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptT [interruptProcedure]↔) Accumulate to the rate of change of the abundancesGas property of an verySimple implementation of the disk component class.

<type(abundances)> abundancesGasRateGet() Get the rate of change of the abundancesGas property of an verySimple implementation of the disk component class.

<void> abundancesGasScale(<type(abundances)> setValue→) Set the absolute scale of the abundancesGas property of an verySimple implementation of the disk component class.

<void> abundancesGasSet(<type(abundances)> setValue→) Set the abundancesGas property of an verySimple implementation of the disk component class.

<type(abundances)> abundancesStellar() Get the abundancesStellar property of an verySimple implementation of the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> abundancesStellarAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the abundancesStellar property

<integer> abundancesStellarCount() Return a count of the number of scalar properties in the abundancesStellar property of an verySimple implementation of the disk component class.

<void> abundancesStellarInactive() Indicate that the abundancesStellar property of an verySimple implementation of the disk component class is inactive for differential equation solving.

<logical> abundancesStellarIsGettable() Returns true if the abundancesStellar property is gettable for the disk component class.

<logical> abundancesStellarIsSettable() Specify whether the abundancesStellar property of the disk component is settable.

<void> abundancesStellarRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interr [interruptProcedure]↔) Accumulate to the rate of change of the abundancesStellar property of an verySimple implementation of the disk component class.

<type(abundances)> abundancesStellarRateGet() Get the rate of change of the abundancesStellar property of an verySimple implementation of the disk component class.

<void> abundancesStellarScale(<type(abundances)> setValue→) Set the absolute scale of the abundancesStellar property of an verySimple implementation of the disk component class.

<void> abundancesStellarSet(<type(abundances)> setValue→) Set the abundancesStellar property of an verySimple implementation of the disk component class.

<double precision> angularMomentum() Returns the default value for the angularMomentum property for the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> angularMomentumAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the angularMomentum property

<integer> angularMomentumCount() Compute the count of evolvable quantities in the angularMomentum property of the DiskStandard component.

<logical> angularMomentumIsGettable() Returns true if the angularMomentum property is gettable for the disk component class.

<logical> angularMomentumIsSettable() Specify whether the angularMomentum property of the disk component is settable.

<void> angularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interrup [interruptProcedure]↔) Accept a rate set for the angularMomentum property of the disk component class. Trigger an interrupt to create the component.

<double precision> angularMomentumRateGet() Returns a zero rate for the angularMomentum property for the disk component class.

<void> angularMomentumScale(<double precision> value) Set the scale of the angularMomentum property of the DiskStandard component.

<void> angularMomentumSet(<double precision> value) Set the angularMomentum property of the disk component.

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> attachPipe() Attach pipes to the very simple disk component.

<void> builder(<*type(node)> componentDefinition→) Build a verySimple implementation of the disk component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a verySimple implementation of the disk component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a verySimple implementation of the disk component from array.

<void> destroy() Finalize a verySimple implementation of the disk component.

<void> dumpASCII() Dump the content of a disk component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<double precision> fractionMassRetained() Returns the default value for the fractionMassRetained property for the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> fractionMassRetainedAttributeMatch(<logic [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the fractionMassRetained property

<integer> fractionMassRetainedCount() Compute the count of evolvable quantities in the fractionMassRetained property of the DiskStandard component.

<logical> fractionMassRetainedIsGettable() Returns true if the fractionMassRetained property is gettable for the disk component class.

<logical> fractionMassRetainedIsSettable() Specify whether the fractionMassRetained property of the disk component is settable.

<void> fractionMassRetainedRate(<double precision> value) Cumulate to the rate of the fractionMassRetained property of the DiskStandard component.

<double precision> fractionMassRetainedRateGet() Returns a zero rate for the fractionMassRetained property for the disk component class.

<void> fractionMassRetainedScale(<double precision> value) Set the scale of the fractionMassRetained property of the DiskStandard component.

<void> fractionMassRetainedSet(<double precision> value) Set the fractionMassRetained property of the disk component.

<double precision> halfMassRadius() Returns the default value for the halfMassRadius property for the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> halfMassRadiusAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the halfMassRadius property

<logical> halfMassRadiusIsGettable() Returns true if the halfMassRadius property is gettable for the disk component class.

<logical> halfMassRadiusIsSettable() Specify whether the halfMassRadius property of the disk component is settable.

<double precision> halfMassRadiusRateGet() Returns a zero rate for the halfMassRadius property for the disk component class.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a verySimple member of the disk component.

<logical> isInitialized() Get the isInitialized property of an verySimple implementation of the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> isInitializedAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the isInitialized property

<logical> isInitializedIsGettable() Returns true if the isInitialized property is gettable for the disk component class.

<logical> isInitializedIsSettable() Specify whether the isInitialized property of the disk component is settable.

<logical> isInitializedRateGet() Returns a zero rate for the isInitialized property for the disk component class.

<void> isInitializedSet(<logical> setValue→) Set the isInitialized property of an verySimple implementation of the disk component class.

<type(stellarLuminosities)> luminositiesStellar() Get the luminositiesStellar property of an verySimple implementation of the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> luminositiesStellarAttributeMatch(<logica [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the luminositiesStellar property

<integer> luminositiesStellarCount() Return a count of the number of scalar properties in the luminositiesStellar property of an verySimple implementation of the disk component class.

<void> luminositiesStellarInactive() Indicate that the luminositiesStellar property of an verySimple implementation of the disk component class is inactive for differential equation solving.

<logical> luminositiesStellarIsGettable() Returns true if the luminositiesStellar property is gettable for the disk component class.

<logical> luminositiesStellarIsSettable() Specify whether the luminositiesStellar property of the disk component is settable.

<void> luminositiesStellarRate(<type(stellarLuminosities)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the luminositiesStellar property of an verySimple implementation of the disk component class.

<type(stellarLuminosities)> luminositiesStellarRateGet() Get the rate of change of the luminositiesStellar property of an verySimple implementation of the disk component class.

<void> luminositiesStellarScale(<type(stellarLuminosities)> setValue→) Set the absolute scale of the luminositiesStellar property of an verySimple implementation of the disk component class.

<void> luminositiesStellarSet(<type(stellarLuminosities)> setValue→) Set the luminositiesStellar property of an verySimple implementation of the disk component class.

<double precision> massGas() Get the massGas property of an verySimple implementation of the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> massGasAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the massGas property

<integer> massGasCount() Return a count of the number of scalar properties in the massGas property of an verySimple implementation of the disk component class.

<void> massGasInactive() Indicate that the massGas property of an verySimple implementation of the disk component class is inactive for differential equation solving.

<logical> massGasIsGettable() Returns true if the massGas property is gettable for the disk component class.

<logical> massGasIsSettable() Specify whether the massGas property of the disk component is settable.

<void> massGasRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the massGas property of an verySimple implementation of the disk component class.

<double precision> massGasRateGet() Get the rate of change of the massGas property of an verySimple implementation of the disk component class.

<void> massGasScale(<double precision> setValue→) Set the absolute scale of the massGas property of an verySimple implementation of the disk component class.

<void> massGasSet(<double precision> setValue→) Set the massGas property of an verySimple implementation of the disk component class.

<double precision> massStellar() Get the massStellar property of an verySimple implementation of the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> massStellarAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the massStellar property

<integer> massStellarCount() Return a count of the number of scalar properties in the massStellar property of an verySimple implementation of the disk component class.

<double precision> massStellarFormed() Returns the default value for the massStellarFormed property for the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> massStellarFormedAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the massStellarFormed property

<integer> massStellarFormedCount() Compute the count of evolvable quantities in the massStellarFormed property of the DiskStandard component.

<logical> massStellarFormedIsGettable() Returns true if the massStellarFormed property is gettable for the disk component class.

<logical> massStellarFormedIsSettable() Specify whether the massStellarFormed property of the disk component is settable.

<void> massStellarFormedRate(<double precision> value) Cumulate to the rate of the massStellarFormed property of the DiskStandard component.

<double precision> massStellarFormedRateGet() Returns a zero rate for the massStellarFormed property for the disk component class.

<void> massStellarFormedScale(<double precision> value) Set the scale of the massStellarFormed property of the DiskStandard component.

<void> massStellarFormedSet(<double precision> value) Set the massStellarFormed property of the disk component.

<void> massStellarInactive() Indicate that the massStellar property of an verySimple implementation of the disk component class is inactive for differential equation solving.

<logical> massStellarIsGettable() Returns true if the massStellar property is gettable for the disk component class.

<logical> massStellarIsSettable() Specify whether the massStellar property of the disk component is settable.

<void> massStellarRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTas [interruptProcedure]↔) Accumulate to the rate of change of the massStellar property of an verySimple implementation of the disk component class.

<double precision> massStellarRateGet() Get the rate of change of the massStellar property of an verySimple implementation of the disk component class.

<void> massStellarScale(<double precision> setValue→) Set the absolute scale of the massStellar property of an verySimple implementation of the disk component class.

<void> massStellarSet(<double precision> setValue→) Set the massStellar property of an verySimple implementation of the disk component class.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a verySimple implementation of the disk component class.

<logical> nullIsActive() Return true if the null implementation of the disk component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a disk component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a verySimple implementation of the disk component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a verySimple implementation of the disk component.

<void> postOutput(<double precision> time→) Perform post-output processing for a verySimple implementation of the disk component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<double precision> radius() Returns the default value for the radius property for the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> radiusAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the radius property

<logical> radiusIsGettable() Returns true if the radius property is gettable for the disk component class.

<logical> radiusIsSettable() Specify whether the radius property of the disk component is settable.

<double precision> radiusRateGet() Returns a zero rate for the radius property for the disk component class.

<void> radiusSet(<double precision> value) Set the radius property of the disk component.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a verySimple implementation of the disk component.

<void> serializeASCII() Serialize the contents of a verySimple implementation of the disk component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a verySimple implementation of the disk component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a verySimple implementation of the disk component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a verySimple implementation of the disk component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a verySimple implementation of the disk component to XML.

<integer(c_size_t)> sizeOf() Return the size in bytes of a verySimple implementation of the disk component.

<logical> standardIsActive() Return true if the standard implementation of the disk component is the active choice.

<type(history)> starFormationHistory() Returns the default value for the starFormationHistory property for the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> starFormationHistoryAttributeMatch(<logic
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the starFormationHistory property

<integer> starFormationHistoryCount() Compute the count of evolvable quantities in the starFormationHistory property of the DiskStandard component.

<logical> starFormationHistoryIsGettable() Returns true if the starFormationHistory property is gettable for the disk component class.

<logical> starFormationHistoryIsSettable() Specify whether the starFormationHistory property of the disk component is settable.

<void> starFormationHistoryRate(<type(history)> value) Cumulate to the rate of the starFormationHistory property of the DiskStandard component.

<type(history)> starFormationHistoryRateGet() Returns a zero rate for the starFormationHistory property for the disk component class.

<void> starFormationHistoryScale(<type(history)> value) Set the scale of the starFormationHistory property of the DiskStandard component.

<void> starFormationHistorySet(<type(history)> value) Set the starFormationHistory property of the disk component.

<double precision> starFormationRate() Get the value of the starFormationRate property of the verySimple implementation of the disk component using a deferred function.

<type(varying_string), allocatable, dimension(:) => matches> starFormationRateAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the starFormationRate property

<void> starFormationRateFunction() Set the function to be used for the get method of the starFormationRate property of the DiskVerySimple component.

<logical> starFormationRateIsAttached() Return true if the deferred function used to get the starFormationRate property of the DiskVerySimple component class has been attached.

<logical> starFormationRateIsGettable() Returns true if the starFormationRate property is gettable for the disk component class.

<logical> starFormationRateIsSettable() Specify whether the starFormationRate property of the disk component is settable.

<double precision> starFormationRateRateGet() Returns a zero rate for the starFormationRate property for the disk component class.

<type(history)> stellarPropertiesHistory() Get the stellarPropertiesHistory property of an verySimple implementation of the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> stellarPropertiesHistoryAttributeMatch(<l [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the stellarPropertiesHistory property

<integer> stellarPropertiesHistoryCount() Return a count of the number of scalar properties in the stellarPropertiesHistory property of an verySimple implementation of the disk component class.

<void> stellarPropertiesHistoryInactive() Indicate that the stellarPropertiesHistory property of an verySimple implementation of the disk component class is inactive for differential equation solving.

<logical> stellarPropertiesHistoryIsGettable() Returns true if the stellarPropertiesHistory property is gettable for the disk component class.

<logical> stellarPropertiesHistoryIsSettable() Specify whether the stellarPropertiesHistory property of the disk component is settable.

<void> stellarPropertiesHistoryRate(<type(history)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the stellarPropertiesHistory property of an verySimple implementation of the disk component class.

<type(history)> stellarPropertiesHistoryRateGet() Get the rate of change of the stellarPropertiesHistory property of an verySimple implementation of the disk component class.

<void> stellarPropertiesHistoryScale(<type(history)> setValue→) Set the absolute scale of the stellarPropertiesHistory property of an verySimple implementation of the disk component class.

<void> stellarPropertiesHistorySet(<type(history)> setValue→) Set the stellarPropertiesHistory property of an verySimple implementation of the disk component class.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<type(varying_string)> type() Returns the type name for the verySimple implementation of the disk component class.

<double precision> velocity() Returns the default value for the velocity property for the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> velocityAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the velocity property

<logical> velocityIsGettable() Returns true if the velocity property is gettable for the disk component class.

<logical> velocityIsSettable() Specify whether the velocity property of the disk component is settable.

<double precision> velocityRateGet() Returns a zero rate for the velocity property for the disk component class.

<void> velocitySet(<double precision> value) Set the velocity property of the disk component.

<logical> verySimpleIsActive() Return true if the verySimple implementation of the disk component is the active choice.

<logical> verySimpleSizeIsActive() Return true if the verySimpleSize implementation of the disk component is the active choice.

nodeComponentDiskVerySimpleSize

<type(abundances)> abundancesGas() Get the abundancesGas property of an verySimple implementation of the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> abundancesGasAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the abundancesGas property

<integer> abundancesGasCount() Return a count of the number of scalar properties in the abundancesGas property of an verySimple implementation of the disk component class.

<void> abundancesGasInactive() Indicate that the abundancesGas property of an verySimple implementation of the disk component class is inactive for differential equation solving.

<logical> abundancesGasIsGettable() Returns true if the abundancesGas property is gettable for the disk component class.

<logical> abundancesGasIsSettable() Specify whether the abundancesGas property of the disk component is settable.

<void> abundancesGasRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptT [interruptProcedure]↔) Accumulate to the rate of change of the abundancesGas property of an verySimple implementation of the disk component class.

<type(abundances)> abundancesGasRateGet() Get the rate of change of the abundancesGas property of an verySimple implementation of the disk component class.

<void> abundancesGasScale(<type(abundances)> setValue→) Set the absolute scale of the abundancesGas property of an verySimple implementation of the disk component class.

<void> abundancesGasSet(<type(abundances)> setValue→) Set the abundancesGas property of an verySimple implementation of the disk component class.

<type(abundances)> abundancesStellar() Get the abundancesStellar property of an verySimple implementation of the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> abundancesStellarAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the abundancesStellar property

<integer> abundancesStellarCount() Return a count of the number of scalar properties in the abundancesStellar property of an verySimple implementation of the disk component class.

<void> abundancesStellarInactive() Indicate that the abundancesStellar property of an verySimple implementation of the disk component class is inactive for differential equation solving.

<logical> abundancesStellarIsGettable() Returns true if the abundancesStellar property is gettable for the disk component class.

<logical> abundancesStellarIsSettable() Specify whether the abundancesStellar property of the disk component is settable.

<void> abundancesStellarRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interr
[interruptProcedure]↔) Accumulate to the rate of change of the abundancesStellar property of an verySimple implementation of the disk component class.

<type(abundances)> abundancesStellarRateGet() Get the rate of change of the abundancesStellar property of an verySimple implementation of the disk component class.

<void> abundancesStellarScale(<type(abundances)> setValue→) Set the absolute scale of the abundancesStellar property of an verySimple implementation of the disk component class.

<void> abundancesStellarSet(<type(abundances)> setValue→) Set the abundancesStellar property of an verySimple implementation of the disk component class.

<double precision> angularMomentum() Returns the default value for the angularMomentum property for the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> angularMomentumAttributeMatch(<logical>
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the disk class that have the desired attributes for the angularMomentum property

<integer> angularMomentumCount() Compute the count of evolvable quantities in the angularMomentum property of the DiskStandard component.

<logical> angularMomentumIsGettable() Returns true if the angularMomentum property is gettable for the disk component class.

<logical> angularMomentumIsSettable() Specify whether the angularMomentum property of the disk component is settable.

<void> angularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interrup
[interruptProcedure]↔) Accept a rate set for the angularMomentum property of the disk component class. Trigger an interrupt to create the component.

<double precision> angularMomentumRateGet() Returns a zero rate for the angularMomentum property for the disk component class.

<void> angularMomentumScale(<double precision> value) Set the scale of the angularMomentum property of the DiskStandard component.

<void> angularMomentumSet(<double precision> value) Set the angularMomentum property of the disk component.

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> attachPipe() Attach pipes to the very simple disk component.

<void> builder(<*type(node)> componentDefinition→) Build a verySimpleSize implementation of the disk component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType>
[massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a verySimpleSize implementation of the disk component from raw (binary) file.

`<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→)` Deserialize evolvable properties of a verySimpleSize implementation of the disk component from array.

`<void> destroy()` Finalize a `verySimpleSize` implementation of the `disk` component.

`<void> dumpASCII()` Dump the content of a `disk` component.

`<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→)` Compute the mass enclosed within a radius.

`<double precision> fractionMassRetained()` Returns the default value for the `fractionMassRetained` property for the `disk` component class.

`<type(varying_string), allocatable, dimension(:) => matches> fractionMassRetainedAttributeMatch(<logic [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)` Return a text list of component implementations in the `disk` class that have the desired attributes for the `fractionMassRetained` property

`<integer> fractionMassRetainedCount()` Compute the count of evolvable quantities in the `fractionMassRetained` property of the `DiskStandard` component.

`<logical> fractionMassRetainedIsGettable()` Returns true if the `fractionMassRetained` property is gettable for the `disk` component class.

`<logical> fractionMassRetainedIsSettable()` Specify whether the `fractionMassRetained` property of the `disk` component is settable.

`<void> fractionMassRetainedRate(<double precision> value)` Cumulate to the rate of the `fractionMassRetained` property of the `DiskStandard` component.

`<double precision> fractionMassRetainedRateGet()` Returns a zero rate for the `fractionMassRetained` property for the `disk` component class.

`<void> fractionMassRetainedScale(<double precision> value)` Set the scale of the `fractionMassRetained` property of the `DiskStandard` component.

`<void> fractionMassRetainedSet(<double precision> value)` Set the `fractionMassRetained` property of the `disk` component.

`<double precision> halfMassRadius()` Returns the default value for the `halfMassRadius` property for the `disk` component class.

`<type(varying_string), allocatable, dimension(:) => matches> halfMassRadiusAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)` Return a text list of component implementations in the `disk` class that have the desired attributes for the `halfMassRadius` property

`<logical> halfMassRadiusIsGettable()` Returns true if the `halfMassRadius` property is gettable for the `disk` component class.

`<logical> halfMassRadiusIsSettable()` Specify whether the `halfMassRadius` property of the `disk` component is settable.

`<double precision> halfMassRadiusRateGet()` Returns a zero rate for the `halfMassRadius` property for the `disk` component class.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a verySimpleSize member of the disk component.

<logical> isInitialized() Get the isInitialized property of an verySimple implementation of the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> isInitializedAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the isInitialized property

<logical> isInitializedIsGettable() Returns true if the isInitialized property is gettable for the disk component class.

<logical> isInitializedIsSettable() Specify whether the isInitialized property of the disk component is settable.

<logical> isInitializedRateGet() Returns a zero rate for the isInitialized property for the disk component class.

<void> isInitializedSet(<logical> setValue→) Set the isInitialized property of an verySimple implementation of the disk component class.

<type(stellarLuminosities)> luminositiesStellar() Get the luminositiesStellar property of an verySimple implementation of the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> luminositiesStellarAttributeMatch(<logica [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the luminositiesStellar property

<integer> luminositiesStellarCount() Return a count of the number of scalar properties in the luminositiesStellar property of an verySimple implementation of the disk component class.

<void> luminositiesStellarInactive() Indicate that the luminositiesStellar property of an verySimple implementation of the disk component class is inactive for differential equation solving.

<logical> luminositiesStellarIsGettable() Returns true if the luminositiesStellar property is gettable for the disk component class.

<logical> luminositiesStellarIsSettable() Specify whether the luminositiesStellar property of the disk component is settable.

<void> luminositiesStellarRate(<type(stellarLuminosities)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the luminositiesStellar property of an verySimple implementation of the disk component class.

<type(stellarLuminosities)> luminositiesStellarRateGet() Get the rate of change of the luminositiesStellar property of an verySimple implementation of the disk component class.

<void> luminositiesStellarScale(<type(stellarLuminosities)> setValue→) Set the absolute scale of the luminositiesStellar property of an verySimple implementation of the disk component class.

<void> luminositiesStellarSet(<type(stellarLuminosities)> setValue→) Set the luminositiesStellar property of an verySimple implementation of the disk component class.

<double precision> massGas() Get the massGas property of an verySimple implementation of the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> massGasAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the massGas property

<integer> massGasCount() Return a count of the number of scalar properties in the massGas property of an verySimple implementation of the disk component class.

<void> massGasInactive() Indicate that the massGas property of an verySimple implementation of the disk component class is inactive for differential equation solving.

<logical> massGasIsGettable() Returns true if the massGas property is gettable for the disk component class.

<logical> massGasIsSettable() Specify whether the massGas property of the disk component is settable.

<void> massGasRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the massGas property of an verySimple implementation of the disk component class.

<double precision> massGasRateGet() Get the rate of change of the massGas property of an verySimple implementation of the disk component class.

<void> massGasScale(<double precision> setValue→) Set the absolute scale of the massGas property of an verySimple implementation of the disk component class.

<void> massGasSet(<double precision> setValue→) Set the massGas property of an verySimple implementation of the disk component class.

<double precision> massStellar() Get the massStellar property of an verySimple implementation of the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> massStellarAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the massStellar property

<integer> massStellarCount() Return a count of the number of scalar properties in the massStellar property of an verySimple implementation of the disk component class.

<double precision> massStellarFormed() Returns the default value for the massStellarFormed property for the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> massStellarFormedAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the massStellarFormed property

<integer> massStellarFormedCount() Compute the count of evolvable quantities in the massStellarFormed property of the DiskStandard component.

<logical> massStellarFormedIsGettable() Returns true if the massStellarFormed property is gettable for the disk component class.

<logical> massStellarFormedIsSettable() Specify whether the massStellarFormed property of the disk component is settable.

<void> massStellarFormedRate(<double precision> value) Cumulate to the rate of the massStellarFormed property of the DiskStandard component.

<double precision> massStellarFormedRateGet() Returns a zero rate for the massStellarFormed property for the disk component class.

<void> massStellarFormedScale(<double precision> value) Set the scale of the massStellarFormed property of the DiskStandard component.

<void> massStellarFormedSet(<double precision> value) Set the massStellarFormed property of the disk component.

<void> massStellarInactive() Indicate that the massStellar property of an verySimple implementation of the disk component class is inactive for differential equation solving.

<logical> massStellarIsGettable() Returns true if the massStellar property is gettable for the disk component class.

<logical> massStellarIsSettable() Specify whether the massStellar property of the disk component is settable.

<void> massStellarRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTas [interruptProcedure]↔) Accumulate to the rate of change of the massStellar property of an verySimple implementation of the disk component class.

<double precision> massStellarRateGet() Get the rate of change of the massStellar property of an verySimple implementation of the disk component class.

<void> massStellarScale(<double precision> setValue→) Set the absolute scale of the massStellar property of an verySimple implementation of the disk component class.

<void> massStellarSet(<double precision> setValue→) Set the massStellar property of an verySimple implementation of the disk component class.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a verySimpleSize implementation of the disk component class.

<logical> nullIsActive() Return true if the null implementation of the disk component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a disk component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a verySimpleSize implementation of the disk component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a verySimpleSize implementation of the disk component.

<void> postOutput(<double precision> time→) Perform post-output processing for a verySimpleSize implementation of the disk component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<double precision> radius() Get the radius property of an verySimpleSize implementation of the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> radiusAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the radius property

<logical> radiusIsGettable() Returns true if the radius property is gettable for the disk component class.

<logical> radiusIsSettable() Specify whether the radius property of the disk component is settable.

<double precision> radiusRateGet() Returns a zero rate for the radius property for the disk component class.

<void> radiusSet(<double precision> setValue→) Set the radius property of an verySimpleSize implementation of the disk component class.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a verySimpleSize implementation of the disk component.

<void> serializeASCII() Serialize the contents of a verySimpleSize implementation of the disk component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a verySimpleSize implementation of the disk component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a verySimpleSize implementation of the disk component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a verySimpleSize implementation of the disk component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a verySimpleSize implementation of the disk component to XML.

<integer(c_size_t)> sizeOf() Return the size in bytes of a verySimpleSize implementation of the disk component.

<logical> standardIsActive() Return true if the standard implementation of the disk component is the active choice.

<type(history)> starFormationHistory() Returns the default value for the starFormationHistory property for the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> starFormationHistoryAttributeMatch(<logic [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the starFormationHistory property

<integer> starFormationHistoryCount() Compute the count of evolvable quantities in the starFormationHistory property of the DiskStandard component.

<logical> starFormationHistoryIsGettable() Returns true if the starFormationHistory property is gettable for the disk component class.

<logical> starFormationHistoryIsSettable() Specify whether the starFormationHistory property of the disk component is settable.

<void> starFormationHistoryRate(<type(history)> value) Cumulate to the rate of the starFormationHistory property of the DiskStandard component.

<type(history)> starFormationHistoryRateGet() Returns a zero rate for the starFormationHistory property for the disk component class.

<void> starFormationHistoryScale(<type(history)> value) Set the scale of the starFormationHistory property of the DiskStandard component.

<void> starFormationHistorySet(<type(history)> value) Set the starFormationHistory property of the disk component.

<double precision> starFormationRate() Get the value of the starFormationRate property of the verySimple implementation of the disk component using a deferred function.

<type(varying_string), allocatable, dimension(:) => matches> starFormationRateAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the starFormationRate property

<void> starFormationRateFunction() Set the function to be used for the get method of the starFormationRate property of the DiskVerySimple component.

<logical> starFormationRateIsAttached() Return true if the deferred function used to get the starFormationRate property of the DiskVerySimple component class has been attached.

<logical> starFormationRateIsGettable() Returns true if the starFormationRate property is gettable for the disk component class.

<logical> starFormationRateIsSettable() Specify whether the starFormationRate property of the disk component is settable.

<double precision> starFormationRateRateGet() Returns a zero rate for the starFormationRate property for the disk component class.

<type(history)> stellarPropertiesHistory() Get the stellarPropertiesHistory property of an verySimple implementation of the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> stellarPropertiesHistoryAttributeMatch(<l [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the stellarPropertiesHistory property

<integer> stellarPropertiesHistoryCount() Return a count of the number of scalar properties in the stellarPropertiesHistory property of an verySimple implementation of the disk component class.

<void> stellarPropertiesHistoryInactive() Indicate that the stellarPropertiesHistory property of an verySimple implementation of the disk component class is inactive for differential equation solving.

<logical> stellarPropertiesHistoryIsGettable() Returns true if the stellarPropertiesHistory property is gettable for the disk component class.

<logical> stellarPropertiesHistoryIsSettable() Specify whether the stellarPropertiesHistory property of the disk component is settable.

<void> stellarPropertiesHistoryRate(<type(history)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the stellarPropertiesHistory property of an verySimple implementation of the disk component class.

<type(history)> stellarPropertiesHistoryRateGet() Get the rate of change of the stellarPropertiesHistory property of an verySimple implementation of the disk component class.

<void> stellarPropertiesHistoryScale(<type(history)> setValue→) Set the absolute scale of the stellarPropertiesHistory property of an verySimple implementation of the disk component class.

<void> stellarPropertiesHistorySet(<type(history)> setValue→) Set the stellarPropertiesHistory property of an verySimple implementation of the disk component class.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<type(varying_string)> type() Returns the type name for the verySimpleSize implementation of the disk component class.

<double precision> velocity() Get the velocity property of an verySimpleSize implementation of the disk component class.

<type(varying_string), allocatable, dimension(:) => matches> velocityAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the disk class that have the desired attributes for the velocity property

<logical> velocityIsGettable() Returns true if the velocity property is gettable for the disk component class.

<logical> velocityIsSettable() Specify whether the velocity property of the disk component is settable.

<double precision> velocityRateGet() Returns a zero rate for the velocity property for the disk component class.

<void> velocitySet(<double precision> setValue→) Set the velocity property of an verySimpleSize implementation of the disk component class.

<logical> verySimpleIsActive() Return true if the verySimple implementation of the disk component is the active choice.

<logical> verySimpleSizeIsActive() Return true if the verySimpleSize implementation of the disk component is the active choice.

nodeComponentDynamicsStatistics

<double precision, dimension(:), allocatable => classDefault> adiabaticRatio() Returns the default value for the adiabaticRatio property for the dynamicsStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> adiabaticRatioAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the dynamicsStatistics class that have the desired attributes for the adiabaticRatio property

<logical> adiabaticRatioIsGettable() Returns true if the adiabaticRatio property is gettable for the dynamicsStatistics component class.

<logical> adiabaticRatioIsSettable() Specify whether the adiabaticRatio property of the dynamicsStatistics component is settable.

<double precision, dimension(:), allocatable => classDefault> adiabaticRatioRateGet() Returns a zero rate for the adiabaticRatio property for the dynamicsStatistics component class.

<void> adiabaticRatioSet(<double precision[:]> value) Set the adiabaticRatio property of the dynamicsStatistics component.

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<double precision, dimension(:), allocatable => classDefault> barInstabilityTimescale() Returns the default value for the barInstabilityTimescale property for the dynamicsStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> barInstabilityTimescaleAttributeMatch(<lo [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the dynamicsStatistics class that have the desired attributes for the barInstabilityTimescale property

<logical> barInstabilityTimescaleIsGettable() Returns true if the barInstabilityTimescale property is gettable for the dynamicsStatistics component class.

<logical> barInstabilityTimescaleIsSettable() Specify whether the barInstabilityTimescale property of the dynamicsStatistics component is settable.

<double precision, dimension(:), allocatable => classDefault> barInstabilityTimescaleRateGet() Returns a zero rate for the barInstabilityTimescale property for the dynamicsStatistics component class.

<void> barInstabilityTimescaleSet(<double precision[:]> value) Set the barInstabilityTimescale property of the dynamicsStatistics component.

<logical> barsIsActive() Return true if the bars implementation of the dynamicsStatistics component is the active choice.

<void> builder(<*type(node)> componentDefinition→) Build a generic dynamicsStatistics component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Read properties from raw file.

<void> deserializeValues(<double(:)> array→, <integer> propertyType→) Deserialize the evolvable quantities from an array.

<void> destroy() Finalize a generic dynamicsStatistics component.

<void> dumpASCII() Dump the content of a dynamicsStatistics component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a generic dynamicsStatistics component.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a nodeComponent object.

<logical> nullIsActive() Return true if the null implementation of the dynamicsStatistics component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a dynamicsStatistics component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a generic dynamicsStatistics component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Establish the names of properties to output for a generic dynamicsStatistics component.

<void> postOutput(<double precision> time→) Perform post-output processing of a dynamicsStatistics component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

`<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→)` Compute the rotation curve gradient.

`<void> serializationOffsets()` Set offsets into serialization arrays.

`<void> serializeASCII()` Serialize the content of a `dynamicsStatistics` component to ASCII.

`<integer> serializeCount(<integer> propertyType→)` Return a count of the number of evolvable quantities to be evolved.

`<void> serializeRaw(<integer> fileHandle→)` Generate a binary dump of all properties.

`<void> serializeValues(<double(:)> array←, <integer> propertyType→)` Serialize the evolvable quantities to an array.

`<void> serializeXML()` Generate an XML dump of all properties.

`<integer(c_size_t)> sizeOf()` Return the size in bytes of a nodeComponentDynamicsStatistics component.

`<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→)` Compute the surface density.

`<double precision, dimension(:), allocatable => classDefault> time()` Returns the default value for the `time` property for the `dynamicsStatistics` component class.

`<type(varying_string), allocatable, dimension(:) => matches> timeAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)` Return a text list of component implementations in the `dynamicsStatistics` class that have the desired attributes for the `time` property

`<logical> timeIsGettable()` Returns true if the `time` property is gettable for the `dynamicsStatistics` component class.

`<logical> timeIsSettable()` Specify whether the `time` property of the `dynamicsStatistics` component is settable.

`<double precision, dimension(:), allocatable => classDefault> timeRateGet()` Returns a zero rate for the `time` property for the `dynamicsStatistics` component class.

`<void> timeSet(<double precision[:]> value)` Set the `time` property of the `dynamicsStatistics` component.

`<type(varying_string)> type()` Returns the type name for the dynamicsStatistics component class.

nodeComponentDynamicsStatisticsBars

`<double precision, dimension(:), allocatable => propertyValue> adiabaticRatio()` Get the `adiabaticRatio` property of an `bars` implementation of the `dynamicsStatistics` component class.

`<type(varying_string), allocatable, dimension(:) => matches> adiabaticRatioAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)` Return a text list of component implementations in the `dynamicsStatistics` class that have the desired attributes for the `adiabaticRatio` property

<logical> adiabaticRatioIsGettable() Returns true if the adiabaticRatio property is gettable for the dynamicsStatistics component class.

<logical> adiabaticRatioIsSettable() Specify whether the adiabaticRatio property of the dynamicsStatistics component is settable.

<double precision, dimension(:), allocatable => classDefault> adiabaticRatioRateGet() Returns a zero rate for the adiabaticRatio property for the dynamicsStatistics component class.

<void> adiabaticRatioSet(<double precision[:]> setValue→) Set the adiabaticRatio property of an bars implementation of the dynamicsStatistics component class.

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<double precision, dimension(:), allocatable => propertyValue> barInstabilityTimescale() Get the barInstabilityTimescale property of an bars implementation of the dynamicsStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> barInstabilityTimescaleAttributeMatch(<lo [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the dynamicsStatistics class that have the desired attributes for the barInstabilityTimescale property

<logical> barInstabilityTimescaleIsGettable() Returns true if the barInstabilityTimescale property is gettable for the dynamicsStatistics component class.

<logical> barInstabilityTimescaleIsSettable() Specify whether the barInstabilityTimescale property of the dynamicsStatistics component is settable.

<double precision, dimension(:), allocatable => classDefault> barInstabilityTimescaleRateGet() Returns a zero rate for the barInstabilityTimescale property for the dynamicsStatistics component class.

<void> barInstabilityTimescaleSet(<double precision[:]> setValue→) Set the barInstabilityTimescale property of an bars implementation of the dynamicsStatistics component class.

<logical> barsIsActive() Return true if the bars implementation of the dynamicsStatistics component is the active choice.

<void> builder(<*type(node)> componentDefinition→) Build a bars implementation of the dynamicsStatistics component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a bars implementation of the dynamicsStatistics component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a bars implementation of the dynamicsStatistics component from array.

<void> destroy() Finalize a bars implementation of the dynamicsStatistics component.

<void> dumpASCII() Dump the content of a dynamicsStatistics component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a bars member of the dynamicsStatistics component.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a bars implementation of the dynamicsStatistics component class.

<logical> nullIsActive() Return true if the null implementation of the dynamicsStatistics component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a dynamicsStatistics component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a bars implementation of the dynamicsStatistics component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a bars implementation of the dynamicsStatistics component.

<void> postOutput(<double precision> time→) Perform post-output processing of a dynamicsStatistics component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<void> record(<double> time→, <double> barInstabilityTimescale→, <double>adiabaticRatio→) Record the current bar-dynamical state of the node.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a bars implementation of the dynamicsStatistics component.

<void> serializeASCII() Serialize the contents of a bars implementation of the dynamicsStatistics component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a bars implementation of the dynamicsStatistics component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a bars implementation of the dynamicsStatistics component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a bars implementation of the dynamicsStatistics component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a bars implementation of the dynamicsStatistics component to XML.

<integer(c_size_t)> sizeOf() Return the size in bytes of a bars implementation of the dynamicsStatistics component.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<double precision, dimension(:), allocatable => propertyValue> time() Get the time property of an bars implementation of the dynamicsStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> timeAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the dynamicsStatistics class that have the desired attributes for the time property

<logical> timeIsGettable() Returns true if the time property is gettable for the dynamicsStatistics component class.

<logical> timeIsSettable() Specify whether the time property of the dynamicsStatistics component is settable.

<double precision, dimension(:), allocatable => classDefault> timeRateGet() Returns a zero rate for the time property for the dynamicsStatistics component class.

<void> timeSet(<double precision[:]> setValue→) Set the time property of an bars implementation of the dynamicsStatistics component class.

<type(varying_string)> type() Returns the type name for the bars implementation of the dynamicsStatistics component class.

nodeComponentDynamicsStatisticsNull

<double precision, dimension(:), allocatable => classDefault> adiabaticRatio() Returns the default value for the adiabaticRatio property for the dynamicsStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> adiabaticRatioAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the dynamicsStatistics class that have the desired attributes for the adiabaticRatio property

<logical> adiabaticRatioIsGettable() Returns true if the adiabaticRatio property is gettable for the dynamicsStatistics component class.

<logical> adiabaticRatioIsSettable() Specify whether the adiabaticRatio property of the dynamicsStatistics component is settable.

<double precision, dimension(:), allocatable => classDefault> adiabaticRatioRateGet() Returns a zero rate for the adiabaticRatio property for the dynamicsStatistics component class.

<void> adiabaticRatioSet(<double precision[:]> value) Set the adiabaticRatio property of the dynamicsStatistics component.

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<double precision, dimension(:), allocatable => classDefault> barInstabilityTimescale() Returns the default value for the barInstabilityTimescale property for the dynamicsStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> barInstabilityTimescaleAttributeMatch(<lo [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the dynamicsStatistics class that have the desired attributes for the barInstabilityTimescale property

<logical> barInstabilityTimescaleIsGettable() Returns true if the barInstabilityTimescale property is gettable for the dynamicsStatistics component class.

<logical> barInstabilityTimescaleIsSettable() Specify whether the barInstabilityTimescale property of the dynamicsStatistics component is settable.

<double precision, dimension(:), allocatable => classDefault> barInstabilityTimescaleRateGet() Returns a zero rate for the barInstabilityTimescale property for the dynamicsStatistics component class.

<void> barInstabilityTimescaleSet(<double precision[:]> value) Set the barInstabilityTimescale property of the dynamicsStatistics component.

<logical> barsIsActive() Return true if the bars implementation of the dynamicsStatistics component is the active choice.

<void> builder(<*type(node)> componentDefinition→) Build a null implementation of the dynamicsStatistics component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a null implementation of the dynamicsStatistics component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a null implementation of the dynamicsStatistics component from array.

<void> destroy() Finalize a null implementation of the dynamicsStatistics component.

<void> dumpASCII() Dump the content of a dynamicsStatistics component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a null member of the dynamicsStatistics component.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a null implementation of the dynamicsStatistics component class.

<logical> nullIsActive() Return true if the null implementation of the dynamicsStatistics component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a dynamicsStatistics component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a null implementation of the dynamicsStatistics component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a null implementation of the dynamicsStatistics component.

<void> postOutput(<double precision> time→) Perform post-output processing of a dynamicsStatistics component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a null implementation of the dynamicsStatistics component.

<void> serializeASCII() Serialize the contents of a null implementation of the dynamicsStatistics component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a null implementation of the dynamicsStatistics component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a null implementation of the dynamicsStatistics component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a null implementation of the dynamicsStatistics component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a null implementation of the dynamicsStatistics component to XML.

<integer(c_size_t)> sizeOf() Return the size in bytes of a null implementation of the dynamicsStatistics component.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<double precision, dimension(:), allocatable => classDefault> time() Returns the default value for the time property for the dynamicsStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> timeAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the dynamicsStatistics class that have the desired attributes for the time property

<logical> timeIsGettable() Returns true if the time property is gettable for the dynamicsStatistics component class.

<logical> timeIsSettable() Specify whether the time property of the dynamicsStatistics component is settable.

<double precision, dimension(:), allocatable => classDefault> timeRateGet() Returns a zero rate for the time property for the dynamicsStatistics component class.

<void> timeSet(<double precision[:]> value) Set the time property of the dynamicsStatistics component.

<type(varying_string)> type() Returns the type name for the null implementation of the dynamicsStatistics component class.

nodeComponentFormationTime

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a generic formationTime component from a supplied XML definition.

<logical> Cole2000IsActive() Return true if the Cole2000 implementation of the formationTime component is the active choice.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Read properties from raw file.

<void> deserializeValues(<double(:)> array→, <integer> propertyType→) Deserialize the evolvable quantities from an array.

<void> destroy() Finalize a generic formationTime component.

<void> dumpASCII() Dump the content of a formationTime component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<double precision> formationTime() Returns the default value for the formationTime property for the formationTime component class.

<type(varying_string), allocatable, dimension(:) => matches> formationTimeAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the formationTime class that have the desired attributes for the formationTime property

<logical> formationTimeIsGettable() Returns true if the formationTime property is gettable for the formationTime component class.

<logical> formationTimeIsSettable() Specify whether the formationTime property of the formationTime component is settable.

<double precision> formationTimeRateGet() Returns a zero rate for the formationTime property for the formationTime component class.

<void> formationTimeSet(<double precision> value) Set the formationTime property of the formationTime component.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a generic formationTime component.

<logical> massFractionIsActive() Return true if the massFraction implementation of the formationTime component is the active choice.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a nodeComponent object.

<logical> nullIsActive() Return true if the null implementation of the formationTime component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a formationTime component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a generic formationTime component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Establish the names of properties to output for a generic formationTime component.

<void> postOutput(<double precision> time→) Perform post-output processing of a formationTime component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets() Set offsets into serialization arrays.

<void> serializeASCII() Serialize the content of a formationTime component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the number of evolvable quantities to be evolved.

<void> serializeRaw(<integer> fileHandle→) Generate a binary dump of all properties.

<void> serializeValues(<double(:)> array←, <integer> propertyType→) Serialize the evolvable quantities to an array.

<void> serializeXML() Generate an XML dump of all properties.

<integer(c_size_t)> sizeOf() Return the size in bytes of a nodeComponentFormationTime component.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<type(varying_string)> type() Returns the type name for the formationTime component class.

nodeComponentFormationTimeCole2000

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a Cole2000 implementation of the formationTime component from a supplied XML definition.

<logical> Cole2000IsActive() Return true if the Cole2000 implementation of the formationTime component is the active choice.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a Cole2000 implementation of the formationTime component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a Cole2000 implementation of the formationTime component from array.

<void> destroy() Finalize a Cole2000 implementation of the formationTime component.

<void> dumpASCII() Dump the content of a formationTime component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<double precision> formationTime() Returns the default value for the formationTime property for the formationTime component class.

<type(varying_string), allocatable, dimension(:) => matches> formationTimeAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the formationTime class that have the desired attributes for the formationTime property

<logical> formationTimeIsGettable() Returns true if the formationTime property is gettable for the formationTime component class.

<logical> formationTimeIsSettable() Specify whether the formationTime property of the formationTime component is settable.

<double precision> formationTimeRateGet() Returns a zero rate for the formationTime property for the formationTime component class.

<void> formationTimeSet(<double precision> value) Set the formationTime property of the formationTime component.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a Cole2000 member of the formationTime component.

<logical> massFractionIsActive() Return true if the massFraction implementation of the formationTime component is the active choice.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a Cole2000 implementation of the formationTime component class.

<logical> nullIsActive() Return true if the null implementation of the formationTime component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a formationTime component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a Cole2000 implementation of the formationTime component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a Cole2000 implementation of the formationTime component.

<void> postOutput(<double precision> time→) Perform post-output processing of a formationTime component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a Cole2000 implementation of the formationTime component.

<void> serializeASCII() Serialize the contents of a Cole2000 implementation of the formationTime component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a Cole2000 implementation of the formationTime component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a Cole2000 implementation of the formationTime component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a Cole2000 implementation of the formationTime component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a Cole2000 implementation of the formationTime component to XML.

<integer(c_size_t)> sizeOf() Return the size in bytes of a Cole2000 implementation of the formationTime component.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<type(varying_string)> type() Returns the type name for the Cole2000 implementation of the formationTime component class.

nodeComponentFormationTimeMassFraction

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a massFraction implementation of the formationTime component from a supplied XML definition.

<logical> Cole2000IsActive() Return true if the Cole2000 implementation of the formationTime component is the active choice.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a massFraction implementation of the formationTime component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a massFraction implementation of the formationTime component from array.

<void> destroy() Finalize a massFraction implementation of the formationTime component.

<void> dumpASCII() Dump the content of a formationTime component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<double precision> formationTime() Get the formationTime property of an massFraction implementation of the formationTime component class.

<type(varying_string), allocatable, dimension(:) => matches> formationTimeAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the formationTime class that have the desired attributes for the formationTime property

<logical> formationTimeIsGettable() Returns true if the formationTime property is gettable for the formationTime component class.

<logical> formationTimeIsSettable() Specify whether the formationTime property of the formationTime component is settable.

<double precision> formationTimeRateGet() Returns a zero rate for the formationTime property for the formationTime component class.

<void> formationTimeSet(<double precision> setValue→) Set the formationTime property of an massFraction implementation of the formationTime component class.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a massFraction member of the formationTime component.

<logical> massFractionIsActive() Return true if the massFraction implementation of the formationTime component is the active choice.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a massFraction implementation of the formationTime component class.

<logical> nullIsActive() Return true if the null implementation of the formationTime component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a formationTime component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a massFraction implementation of the formationTime component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a massFraction implementation of the formationTime component.

<void> postOutput(<double precision> time→) Perform post-output processing of a formationTime component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a massFraction implementation of the formationTime component.

<void> serializeASCII() Serialize the contents of a massFraction implementation of the formationTime component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a massFraction implementation of the formationTime component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a massFraction implementation of the formationTime component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a massFraction implementation of the formationTime component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a massFraction implementation of the formationTime component to XML.

<integer(c_size_t)> sizeOf() Return the size in bytes of a massFraction implementation of the formationTime component.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<type(varying_string)> type() Returns the type name for the massFraction implementation of the formationTime component class.

nodeComponentFormationTimeNull

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a null implementation of the formationTime component from a supplied XML definition.

<logical> Cole2000IsActive() Return true if the Cole2000 implementation of the formationTime component is the active choice.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a null implementation of the formationTime component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a null implementation of the formationTime component from array.

<void> destroy() Finalize a null implementation of the formationTime component.

<void> dumpASCII() Dump the content of a formationTime component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<double precision> formationTime() Returns the default value for the formationTime property for the formationTime component class.

<type(varying_string), allocatable, dimension(:) => matches> formationTimeAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the formationTime class that have the desired attributes for the formationTime property

<logical> formationTimeIsGettable() Returns true if the formationTime property is gettable for the formationTime component class.

<logical> formationTimeIsSettable() Specify whether the formationTime property of the formationTime component is settable.

<double precision> formationTimeRateGet() Returns a zero rate for the formationTime property for the formationTime component class.

<void> formationTimeSet(<double precision> value) Set the formationTime property of the formationTime component.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a null member of the formationTime component.

<logical> massFractionIsActive() Return true if the massFraction implementation of the formationTime component is the active choice.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a null implementation of the formationTime component class.

<logical> nullIsActive() Return true if the null implementation of the formationTime component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a formationTime component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a null implementation of the formationTime component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a null implementation of the formationTime component.

<void> postOutput(<double precision> time→) Perform post-output processing of a formationTime component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a null implementation of the formationTime component.

<void> serializeASCII() Serialize the contents of a null implementation of the formationTime component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a null implementation of the formationTime component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a null implementation of the formationTime component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a null implementation of the formationTime component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a null implementation of the formationTime component to XML.

<integer(c_size_t)> sizeOf() Return the size in bytes of a null implementation of the formationTime component.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<type(varying_string)> type() Returns the type name for the null implementation of the formationTime component class.

nodeComponentHostHistory

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a generic hostHistory component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Read properties from raw file.

<void> deserializeValues(<double(:)> array→, <integer> propertyType→) Deserialize the evolvable quantities from an array.

<void> destroy() Finalize a generic hostHistory component.

<void> dumpASCII() Dump the content of a hostHistory component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<double precision> hostMassMaximum() Returns the default value for the hostMassMaximum property for the hostHistory component class.

<type(varying_string), allocatable, dimension(:) => matches> hostMassMaximumAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hostHistory class that have the desired attributes for the hostMassMaximum property

<logical> hostMassMaximumIsGettable() Returns true if the hostMassMaximum property is gettable for the hostHistory component class.

<logical> hostMassMaximumIsSettable() Specify whether the hostMassMaximum property of the hostHistory component is settable.

<double precision> hostMassMaximumRateGet() Returns a zero rate for the hostMassMaximum property for the hostHistory component class.

<void> hostMassMaximumSet(<double precision> value) Set the hostMassMaximum property of the hostHistory component.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter
[interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a generic hostHistory component.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a nodeComponent object.

<logical> nullIsActive() Return true if the null implementation of the hostHistory component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_-
int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔,
<double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)>
outputInstance→, <integer> instance→) Populate output buffers with properties to output for a hostHistory component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double
precision> time→, <integer> instance→) Increment the count of properties to output for a generic hostHistory component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔,
<character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔,
<integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]>
doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision>
time→, <integer> instance→) Establish the names of properties to output for a generic hostHistory component.

<void> postOutput(<double precision> time→) Perform post-output processing of a hostHistory component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→)
Compute the gravitational potential.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType>
[massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType>
[massType]→) Compute the rotation curve gradient.

<void> serializationOffsets() Set offsets into serialization arrays.

<void> serializeASCII() Serialize the content of a hostHistory component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the number of evolvable quantities to be evolved.

<void> serializeRaw(<integer> fileHandle→) Generate a binary dump of all properties.

<void> serializeValues(<double(:)> array←, <integer> propertyType→) Serialize the evolvable quantities to an array.

<void> serializeXML() Generate an XML dump of all properties.

<integer(c_size_t)> sizeOf() Return the size in bytes of a nodeComponentHostHistory component.

<logical> standardIsActive() Return true if the standard implementation of the hostHistory component is the active choice.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<type(varying_string)> type() Returns the type name for the hostHistory component class.

nodeComponentHostHistoryNull

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a null implementation of the hostHistory component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a null implementation of the hostHistory component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a null implementation of the hostHistory component from array.

<void> destroy() Finalize a null implementation of the hostHistory component.

<void> dumpASCII() Dump the content of a hostHistory component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<double precision> hostMassMaximum() Returns the default value for the hostMassMaximum property for the hostHistory component class.

<type(varying_string), allocatable, dimension(:) => matches> hostMassMaximumAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hostHistory class that have the desired attributes for the hostMassMaximum property

<logical> hostMassMaximumIsGettable() Returns true if the hostMassMaximum property is gettable for the hostHistory component class.

<logical> hostMassMaximumIsSettable() Specify whether the hostMassMaximum property of the hostHistory component is settable.

<double precision> hostMassMaximumRateGet() Returns a zero rate for the hostMassMaximum property for the hostHistory component class.

<void> hostMassMaximumSet(<double precision> value) Set the hostMassMaximum property of the hostHistory component.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a null member of the hostHistory component.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a null implementation of the hostHistory component class.

<logical> nullIsActive() Return true if the null implementation of the hostHistory component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a hostHistory component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a null implementation of the hostHistory component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a null implementation of the hostHistory component.

<void> postOutput(<double precision> time→) Perform post-output processing of a hostHistory component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a null implementation of the hostHistory component.

<void> serializeASCII() Serialize the contents of a null implementation of the hostHistory component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a null implementation of the hostHistory component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a null implementation of the hostHistory component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a null implementation of the hostHistory component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a null implementation of the hostHistory component to XML.

<integer(c_size_t)> sizeOf() Return the size in bytes of a null implementation of the hostHistory component.

<logical> standardIsActive() Return true if the standard implementation of the hostHistory component is the active choice.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<type(varying_string)> type() Returns the type name for the null implementation of the hostHistory component class.

nodeComponentHostHistoryStandard

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a standard implementation of the hostHistory component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a standard implementation of the hostHistory component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a standard implementation of the hostHistory component from array.

<void> destroy() Finalize a standard implementation of the hostHistory component.

<void> dumpASCII() Dump the content of a hostHistory component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<double precision> hostMassMaximum() Get the hostMassMaximum property of an standard implementation of the hostHistory component class.

<type(varying_string), allocatable, dimension(:) => matches> hostMassMaximumAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hostHistory class that have the desired attributes for the hostMassMaximum property

<logical> hostMassMaximumIsGettable() Returns true if the hostMassMaximum property is gettable for the hostHistory component class.

<logical> hostMassMaximumIsSettable() Specify whether the hostMassMaximum property of the hostHistory component is settable.

<double precision> hostMassMaximumRateGet() Returns a zero rate for the hostMassMaximum property for the hostHistory component class.

<void> hostMassMaximumSet(<double precision> setValue→) Set the hostMassMaximum property of an standard implementation of the hostHistory component class.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a standard member of the hostHistory component.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a standard implementation of the hostHistory component class.

<logical> nullIsActive() Return true if the null implementation of the hostHistory component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a hostHistory component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a standard implementation of the hostHistory component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a standard implementation of the hostHistory component.

<void> postOutput(<double precision> time→) Perform post-output processing of a hostHistory component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a standard implementation of the hostHistory component.

<void> serializeASCII() Serialize the contents of a standard implementation of the hostHistory component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a standard implementation of the hostHistory component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a standard implementation of the hostHistory component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a standard implementation of the hostHistory component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a standard implementation of the hostHistory component to XML.

<integer(c_size_t)> sizeOf() Return the size in bytes of a standard implementation of the hostHistory component.

<logical> standardIsActive() Return true if the standard implementation of the hostHistory component is the active choice.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<type(varying_string)> type() Returns the type name for the standard implementation of the hostHistory component class.

nodeComponentHotHalo

<type(abundances)> abundances() Returns the default value for the abundances property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> abundancesAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the abundances property

<type(abundances)> abundancesCold() Returns the default value for the abundancesCold property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> abundancesColdAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the abundancesCold property

<integer> abundancesColdCount() Compute the count of evolvable quantities in the abundancesCold property of the HotHaloColdMode component.

<logical> abundancesColdIsGettable() Returns true if the abundancesCold property is gettable for the hotHalo component class.

<logical> abundancesColdIsSettable() Specify whether the abundancesCold property of the hotHalo component is settable.

<void> abundancesColdRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interrupt [interruptProcedure]↔) Accept a rate set for the abundancesCold property of the hotHalo component class. Trigger an interrupt to create the component.

<type(abundances)> abundancesColdRateGet() Returns a zero rate for the abundancesCold property for the hotHalo component class.

<void> abundancesColdScale(<type(abundances)> value) Set the scale of the abundancesCold property of the HotHaloColdMode component.

<void> abundancesColdSet(<type(abundances)> value) Set the abundancesCold property of the hotHalo component.

<integer> abundancesCount() Compute the count of evolvable quantities in the abundances property of the HotHaloStandard component.

<logical> abundancesIsGettable() Returns true if the abundances property is gettable for the hotHalo component class.

<logical> abundancesIsSettable() Specify whether the abundances property of the hotHalo component is settable.

<void> abundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask
[interruptProcedure]↔) Accept a rate set for the abundances property of the hotHalo component class. Trigger an interrupt to create the component.

<type(abundances)> abundancesRateGet() Returns a zero rate for the abundances property for the hotHalo component class.

<void> abundancesScale(<type(abundances)> value) Set the scale of the abundances property of the HotHaloStandard component.

<void> abundancesSet(<type(abundances)> value) Set the abundances property of the hotHalo component.

<double precision> angularMomentum() Returns the default value for the angularMomentum property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> angularMomentumAttributeMatch(<logical>
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the hotHalo class that have the desired attributes for the angularMomentum property

<double precision> angularMomentumCold() Returns the default value for the angularMomentumCold property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> angularMomentumColdAttributeMatch(<logica
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the hotHalo class that have the desired attributes for the angularMomentumCold property

<integer> angularMomentumColdCount() Compute the count of evolvable quantities in the angularMomentumCold property of the HotHaloColdMode component.

<logical> angularMomentumColdIsGettable() Returns true if the angularMomentumCold property is gettable for the hotHalo component class.

<logical> angularMomentumColdIsSettable() Specify whether the angularMomentumCold property of the hotHalo component is settable.

<void> angularMomentumColdRate(<double precision> setValue→, <logical> [interrupt]↔,
<*procedure(interruptTask)> [interruptProcedure]↔) Accept a rate set for the angularMomentumCold property of the hotHalo component class. Trigger an interrupt to create the component.

<double precision> angularMomentumColdRateGet() Returns a zero rate for the angularMomentumCold property for the hotHalo component class.

<void> angularMomentumColdScale(<double precision> value) Set the scale of the angularMomentumCold property of the HotHaloColdMode component.

<void> angularMomentumColdSet(<double precision> value) Set the angularMomentumCold property of the hotHalo component.

<integer> angularMomentumCount() Compute the count of evolvable quantities in the angularMomentum property of the HotHaloStandard component.

<logical> angularMomentumIsGettable() Returns true if the angularMomentum property is gettable for the hotHalo component class.

<logical> angularMomentumIsSettable() Specify whether the angularMomentum property of the hotHalo component is settable.

<void> angularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interrup [interruptProcedure]↔) Accept a rate set for the angularMomentum property of the hotHalo component class. Trigger an interrupt to create the component.

<double precision> angularMomentumRateGet() Returns a zero rate for the angularMomentum property for the hotHalo component class.

<void> angularMomentumScale(<double precision> value) Set the scale of the angularMomentum property of the HotHaloStandard component.

<void> angularMomentumSet(<double precision> value) Set the angularMomentum property of the hotHalo component.

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a generic hotHalo component from a supplied XML definition.

<type(chemicalAbundances)> chemicals() Returns the default value for the chemicals property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> chemicalsAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the chemicals property

<integer> chemicalsCount() Compute the count of evolvable quantities in the chemicals property of the HotHaloStandard component.

<logical> chemicalsIsGettable() Returns true if the chemicals property is gettable for the hotHalo component class.

<logical> chemicalsIsSettable() Specify whether the chemicals property of the hotHalo component is settable.

<void> chemicalsRate(<type(chemicalAbundances)> setValue→, <logical> [interrupt]↔, <*procedure(interr [interruptProcedure]↔) Accept a rate set for the chemicals property of the hotHalo component class. Trigger an interrupt to create the component.

<type(chemicalAbundances)> chemicalsRateGet() Returns a zero rate for the chemicals property for the hotHalo component class.

<void> chemicalsScale(<type(chemicalAbundances)> value) Set the scale of the chemicals property of the HotHaloStandard component.

<void> chemicalsSet(<type(chemicalAbundances)> value) Set the chemicals property of the hotHalo component.

<logical> coldModeIsActive() Return true if the coldMode implementation of the hotHalo component is the active choice.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Read properties from raw file.

<void> deserializeValues(<double(:)> array→, <integer> propertyType→) Deserialize the evolvable quantities from an array.

<void> destroy() Finalize a generic hotHalo component.

<void> dumpASCII() Dump the content of a hotHalo component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<double precision> heatSource() Returns the default value for the heatSource property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> heatSourceAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the heatSource property

<integer> heatSourceCount() Compute the count of evolvable quantities in the heatSource property of the HotHaloStandard component.

<logical> heatSourceIsGettable() Returns true if the heatSource property is gettable for the hotHalo component class.

<logical> heatSourceIsSettable() Specify whether the heatSource property of the hotHalo component is settable.

<void> heatSourceRate(<double precision> value) Cumulate to the rate of the heatSource property of the HotHaloStandard component.

<double precision> heatSourceRateGet() Returns a zero rate for the heatSource property for the hotHalo component class.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<type(abundances)> hotHaloCoolingAbundances() Returns the default value for the hotHaloCoolingAbundances property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> hotHaloCoolingAbundancesAttributeMatch(<l [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the hotHaloCoolingAbundances property

<integer> hotHaloCoolingAbundancesCount() Compute the count of evolvable quantities in the hotHaloCoolingAbundan property of the HotHaloStandard component.

<logical> hotHaloCoolingAbundancesIsGettable() Returns true if the hotHaloCoolingAbundances property is gettable for the hotHalo component class.

<logical> hotHaloCoolingAbundancesIsSettable() Specify whether the hotHaloCoolingAbundances property of the hotHalo component is settable.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAbundancesRateFunction() Set the function to be used for the rate method of the hotHaloCoolingAbundances property of the hotHalo component.

<type(abundances)> hotHaloCoolingAbundancesRateGet() Returns a zero rate for the hotHaloCoolingAbundances property for the hotHalo component class.

<logical> hotHaloCoolingAbundancesRateIsAttached() Return true if the deferred function used to rate the hotHaloCoolingAbundances property of the hotHalo component class has been attached.

<double precision> hotHaloCoolingAngularMomentum() Returns the default value for the hotHaloCoolingAngularMome property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> hotHaloCoolingAngularMomentumAttributeMat [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the hotHaloCoolingAngularMomentum property

<integer> hotHaloCoolingAngularMomentumCount() Compute the count of evolvable quantities in the hotHaloCoolingAngularMomentum property of the HotHaloStandard component.

<logical> hotHaloCoolingAngularMomentumIsGettable() Returns true if the hotHaloCoolingAngularMomentum property is gettable for the hotHalo component class.

<logical> hotHaloCoolingAngularMomentumIsSettable() Specify whether the hotHaloCoolingAngularMomentum property of the hotHalo component is settable.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRateFunction() Set the function to be used for the rate method of the hotHaloCoolingAngularMomentum property of the hotHalo component.

<double precision> hotHaloCoolingAngularMomentumRateGet() Returns a zero rate for the hotHaloCoolingAngularMo property for the hotHalo component class.

<logical> hotHaloCoolingAngularMomentumRateIsAttached() Return true if the deferred function used to rate the hotHaloCoolingAngularMomentum property of the hotHalo component class has been attached.

<double precision> hotHaloCoolingMass() Returns the default value for the hotHaloCoolingMass property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> hotHaloCoolingMassAttributeMatch(<logical [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the hotHaloCoolingMass property

<integer> hotHaloCoolingMassCount() Compute the count of evolvable quantities in the hotHaloCoolingMass property of the HotHaloStandard component.

<logical> hotHaloCoolingMassIsGettable() Returns true if the hotHaloCoolingMass property is gettable for the hotHalo component class.

<logical> hotHaloCoolingMassIsSettable() Specify whether the hotHaloCoolingMass property of the hotHalo component is settable.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRateFunction() Set the function to be used for the rate method of the hotHaloCoolingMass property of the hotHalo component.

<double precision> hotHaloCoolingMassRateGet() Returns a zero rate for the hotHaloCoolingMass property for the hotHalo component class.

<logical> hotHaloCoolingMassRateIsAttached() Return true if the deferred function used to rate the hotHaloCoolingMass property of the hotHalo component class has been attached.

<void> initialize() Initialize a generic hotHalo component.

<logical> isInitialized() Returns the default value for the isInitialized property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> isInitializedAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the isInitialized property

<logical> isInitializedIsGettable() Returns true if the isInitialized property is gettable for the hotHalo component class.

<logical> isInitializedIsSettable() Specify whether the isInitialized property of the hotHalo component is settable.

<logical> isInitializedRateGet() Returns a zero rate for the isInitialized property for the hotHalo component class.

<void> isInitializedSet(<logical> value) Set the isInitialized property of the hotHalo component.

<double precision> mass() Returns the default value for the mass property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> massAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the mass property

<double precision> massCold() Returns the default value for the massCold property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> massColdAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the massCold property

<integer> massColdCount() Compute the count of evolvable quantities in the massCold property of the HotHaloColdMode component.

<logical> massColdIsGettable() Returns true if the massCold property is gettable for the hotHalo component class.

<logical> massColdIsSettable() Specify whether the massCold property of the hotHalo component is settable.

<void> massColdRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accept a rate set for the massCold property of the hotHalo component class. Trigger an interrupt to create the component.

<double precision> massColdRateGet() Returns a zero rate for the massCold property for the hotHalo component class.

<void> massColdScale(<double precision> value) Set the scale of the massCold property of the HotHaloColdMode component.

<void> massColdSet(<double precision> value) Set the massCold property of the hotHalo component.

<integer> massCount() Compute the count of evolvable quantities in the mass property of the HotHaloStandard component.

<logical> massIsGettable() Returns true if the mass property is gettable for the hotHalo component class.

<logical> massIsSettable() Specify whether the mass property of the hotHalo component is settable.

<void> massRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accept a rate set for the mass property of the hotHalo component class. Trigger an interrupt to create the component.

<double precision> massRateGet() Returns a zero rate for the mass property for the hotHalo component class.

<void> massScale(<double precision> value) Set the scale of the mass property of the HotHaloStandard component.

<void> massSet(<double precision> value) Set the mass property of the hotHalo component.

<double precision> massSink() Returns the default value for the massSink property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> massSinkAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the massSink property

<integer> massSinkCount() Compute the count of evolvable quantities in the massSink property of the HotHaloStandard component.

<logical> massSinkIsGettable() Returns true if the massSink property is gettable for the hotHalo component class.

<logical> massSinkIsSettable() Specify whether the massSink property of the hotHalo component is settable.

<void> massSinkRate(<double precision> value) Cumulate to the rate of the massSink property of the HotHaloStandard component.

<double precision> massSinkRateGet() Returns a zero rate for the massSink property for the hotHalo component class.

<double precision> massTotal() Returns the default value for the massTotal property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> massTotalAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the massTotal property

<logical> massTotalIsGettable() Returns true if the massTotal property is gettable for the hotHalo component class.

<logical> massTotalIsSettable() Specify whether the massTotal property of the hotHalo component is settable.

<double precision> massTotalRateGet() Returns a zero rate for the massTotal property for the hotHalo component class.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a nodeComponent object.

<logical> nullIsActive() Return true if the null implementation of the hotHalo component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<double precision> outerRadius() Returns the default value for the outerRadius property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> outerRadiusAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the outerRadius property

<integer> outerRadiusCount() Compute the count of evolvable quantities in the outerRadius property of the HotHaloStandard component.

<logical> outerRadiusIsGettable() Returns true if the outerRadius property is gettable for the hotHalo component class.

<logical> outerRadiusIsSettable() Specify whether the outerRadius property of the hotHalo component is settable.

<void> outerRadiusRate(<double precision> value) Cumulate to the rate of the outerRadius property of the HotHaloStandard component.

<double precision> outerRadiusRateGet() Returns a zero rate for the outerRadius property for the hotHalo component class.

<void> outerRadiusScale(<double precision> value) Set the scale of the outerRadius property of the HotHaloStandard component.

<void> outerRadiusSet(<double precision> value) Set the outerRadius property of the hotHalo component.

<type(abundances)> outflowedAbundances() Returns the default value for the outflowedAbundances property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> outflowedAbundancesAttributeMatch(<logica
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the hotHalo class that have the desired attributes for the outflowedAbundances property

<integer> outflowedAbundancesCount() Compute the count of evolvable quantities in the outflowedAbundances property of the HotHaloStandard component.

<logical> outflowedAbundancesIsGettable() Returns true if the outflowedAbundances property is gettable for the hotHalo component class.

<logical> outflowedAbundancesIsSettable() Specify whether the outflowedAbundances property of the hotHalo component is settable.

<void> outflowedAbundancesRate(<type(abundances)> value) Cumulate to the rate of the outflowedAbundances property of the HotHaloStandard component.

<type(abundances)> outflowedAbundancesRateGet() Returns a zero rate for the outflowedAbundances property for the hotHalo component class.

<void> outflowedAbundancesScale(<type(abundances)> value) Set the scale of the outflowedAbundances property of the HotHaloStandard component.

<void> outflowedAbundancesSet(<type(abundances)> value) Set the outflowedAbundances property of the hotHalo component.

<double precision> outflowedAngularMomentum() Returns the default value for the outflowedAngularMomentum property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> outflowedAngularMomentumAttributeMatch(<l
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the hotHalo class that have the desired attributes for the outflowedAngularMomentum property

<integer> outflowedAngularMomentumCount() Compute the count of evolvable quantities in the outflowedAngularMomen
property of the HotHaloStandard component.

<logical> outflowedAngularMomentumIsGettable() Returns true if the outflowedAngularMomentum property is gettable for the hotHalo component class.

<logical> outflowedAngularMomentumIsSettable() Specify whether the outflowedAngularMomentum property of the hotHalo component is settable.

<void> outflowedAngularMomentumRate(<double precision> value) Cumulate to the rate of the outflowedAngularMomentum property of the HotHaloStandard component.

<double precision> outflowedAngularMomentumRateGet() Returns a zero rate for the outflowedAngularMomentum property for the hotHalo component class.

<void> outflowedAngularMomentumScale(<double precision> value) Set the scale of the outflowedAngularMomentum property of the HotHaloStandard component.

<void> outflowedAngularMomentumSet(<double precision> value) Set the outflowedAngularMomentum property of the hotHalo component.

<double precision> outflowedMass() Returns the default value for the outflowedMass property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> outflowedMassAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the outflowedMass property

<integer> outflowedMassCount() Compute the count of evolvable quantities in the outflowedMass property of the HotHaloStandard component.

<logical> outflowedMassIsGettable() Returns true if the outflowedMass property is gettable for the hotHalo component class.

<logical> outflowedMassIsSettable() Specify whether the outflowedMass property of the hotHalo component is settable.

<void> outflowedMassRate(<double precision> value) Cumulate to the rate of the outflowedMass property of the HotHaloStandard component.

<double precision> outflowedMassRateGet() Returns a zero rate for the outflowedMass property for the hotHalo component class.

<void> outflowedMassScale(<double precision> value) Set the scale of the outflowedMass property of the HotHaloStandard component.

<void> outflowedMassSet(<double precision> value) Set the outflowedMass property of the hotHalo component.

<type(abundances)> outflowingAbundances() Returns the default value for the outflowingAbundances property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> outflowingAbundancesAttributeMatch(<logic
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the outflowingAbundances property

<integer> outflowingAbundancesCount() Compute the count of evolvable quantities in the outflowingAbundances property of the HotHaloStandard component.

<logical> outflowingAbundancesIsGettable() Returns true if the outflowingAbundances property is gettable for the hotHalo component class.

<logical> outflowingAbundancesIsSettable() Specify whether the outflowingAbundances property of the hotHalo component is settable.

<void> outflowingAbundancesRate(<type(abundances)> value) Cumulate to the rate of the outflowingAbundances property of the HotHaloStandard component.

<type(abundances)> outflowingAbundancesRateGet() Returns a zero rate for the outflowingAbundances property for the hotHalo component class.

<double precision> outflowingAngularMomentum() Returns the default value for the outflowingAngularMomentum property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> outflowingAngularMomentumAttributeMatch(<
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the hotHalo class that have the desired attributes for the outflowingAngularMomentum property

<integer> outflowingAngularMomentumCount() Compute the count of evolvable quantities in the outflowingAngularMomentum property of the HotHaloStandard component.

<logical> outflowingAngularMomentumIsGettable() Returns true if the outflowingAngularMomentum property is gettable for the hotHalo component class.

<logical> outflowingAngularMomentumIsSettable() Specify whether the outflowingAngularMomentum property of the hotHalo component is settable.

<void> outflowingAngularMomentumRate(<double precision> value) Cumulate to the rate of the outflowingAngularMomentum property of the HotHaloStandard component.

<double precision> outflowingAngularMomentumRateGet() Returns a zero rate for the outflowingAngularMomentum property for the hotHalo component class.

<double precision> outflowingMass() Returns the default value for the outflowingMass property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> outflowingMassAttributeMatch(<logical>
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the hotHalo class that have the desired attributes for the outflowingMass property

<integer> outflowingMassCount() Compute the count of evolvable quantities in the outflowingMass property of the HotHaloStandard component.

<logical> outflowingMassIsGettable() Returns true if the outflowingMass property is gettable for the hotHalo component class.

<logical> outflowingMassIsSettable() Specify whether the outflowingMass property of the hotHalo component is settable.

<void> outflowingMassRate(<double precision> value) Cumulate to the rate of the outflowingMass property of the HotHaloStandard component.

<double precision> outflowingMassRateGet() Returns a zero rate for the outflowingMass property for the hotHalo component class.

<logical> outflowTrackingIsActive() Return true if the outflowTracking implementation of the hotHalo component is the active choice.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a hotHalo component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a generic hotHalo component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Establish the names of properties to output for a generic hotHalo component.

<void> postOutput(<double precision> time→) Perform post-output processing of a hotHalo component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets() Set offsets into serialization arrays.

<void> serializeASCII() Serialize the content of a hotHalo component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the number of evolvable quantities to be evolved.

<void> serializeRaw(<integer> fileHandle→) Generate a binary dump of all properties.

<void> serializeValues(<double(:)> array←, <integer> propertyType→) Serialize the evolvable quantities to an array.

<void> serializeXML() Generate an XML dump of all properties.

<integer(c_size_t)> sizeOf() Return the size in bytes of a nodeComponentHotHalo component.

<logical> standardIsActive() Return true if the standard implementation of the hotHalo component is the active choice.

<type(abundances)> strippedAbundances() Returns the default value for the strippedAbundances property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> strippedAbundancesAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the strippedAbundances property

<integer> strippedAbundancesCount() Compute the count of evolvable quantities in the strippedAbundances property of the HotHaloStandard component.

<logical> strippedAbundancesIsGettable() Returns true if the strippedAbundances property is gettable for the hotHalo component class.

<logical> strippedAbundancesIsSettable() Specify whether the strippedAbundances property of the hotHalo component is settable.

<void> strippedAbundancesRate(<type(abundances)> value) Cumulate to the rate of the strippedAbundances property of the HotHaloStandard component.

<type(abundances)> strippedAbundancesRateGet() Returns a zero rate for the strippedAbundances property for the hotHalo component class.

<void> strippedAbundancesScale(<type(abundances)> value) Set the scale of the strippedAbundances property of the HotHaloStandard component.

<void> strippedAbundancesSet(<type(abundances)> value) Set the strippedAbundances property of the hotHalo component.

<double precision> strippedMass() Returns the default value for the strippedMass property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> strippedMassAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the strippedMass property

<integer> strippedMassCount() Compute the count of evolvable quantities in the strippedMass property of the HotHaloStandard component.

<logical> strippedMassIsGettable() Returns true if the strippedMass property is gettable for the hotHalo component class.

<logical> strippedMassIsSettable() Specify whether the strippedMass property of the hotHalo component is settable.

<void> strippedMassRate(<double precision> value) Cumulate to the rate of the strippedMass property of the HotHaloStandard component.

<double precision> strippedMassRateGet() Returns a zero rate for the strippedMass property for the hotHalo component class.

<void> strippedMassScale(<double precision> value) Set the scale of the strippedMass property of the HotHaloStandard component.

<void> strippedMassSet(<double precision> value) Set the strippedMass property of the hotHalo component.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<type(abundances)> trackedOutflowAbundances() Returns the default value for the trackedOutflowAbundances property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> trackedOutflowAbundancesAttributeMatch(<l [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the trackedOutflowAbundances property

<integer> trackedOutflowAbundancesCount() Compute the count of evolvable quantities in the trackedOutflowAbundan property of the HotHaloOutflowTracking component.

<logical> trackedOutflowAbundancesIsGettable() Returns true if the trackedOutflowAbundances property is gettable for the hotHalo component class.

<logical> trackedOutflowAbundancesIsSettable() Specify whether the trackedOutflowAbundances property of the hotHalo component is settable.

<void> trackedOutflowAbundancesRate(<type(abundances)> value) Cumulate to the rate of the trackedOutflowAbundances property of the HotHaloOutflowTracking component.

<type(abundances)> trackedOutflowAbundancesRateGet() Returns a zero rate for the trackedOutflowAbundances property for the hotHalo component class.

<void> trackedOutflowAbundancesScale(<type(abundances)> value) Set the scale of the trackedOutflowAbundances property of the HotHaloOutflowTracking component.

<void> trackedOutflowAbundancesSet(<type(abundances)> value) Set the trackedOutflowAbundances property of the hotHalo component.

<double precision> trackedOutflowMass() Returns the default value for the trackedOutflowMass property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> trackedOutflowMassAttributeMatch(<logical [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the trackedOutflowMass property

<integer> trackedOutflowMassCount() Compute the count of evolvable quantities in the trackedOutflowMass property of the HotHaloOutflowTracking component.

<logical> trackedOutflowMassIsGettable() Returns true if the trackedOutflowMass property is gettable for the hotHalo component class.

<logical> trackedOutflowMassIsSettable() Specify whether the trackedOutflowMass property of the hotHalo component is settable.

<void> trackedOutflowMassRate(<double precision> value) Cumulate to the rate of the trackedOutflowMass property of the HotHaloOutflowTracking component.

<double precision> trackedOutflowMassRateGet() Returns a zero rate for the trackedOutflowMass property for the hotHalo component class.

<void> trackedOutflowMassScale(<double precision> value) Set the scale of the trackedOutflowMass property of the HotHaloOutflowTracking component.

<void> trackedOutflowMassSet(<double precision> value) Set the trackedOutflowMass property of the hotHalo component.

<type(varying_string)> type() Returns the type name for the hotHalo component class.

<double precision> unaccretedMass() Returns the default value for the unaccretedMass property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> unaccretedMassAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the unaccretedMass property

<integer> unaccretedMassCount() Compute the count of evolvable quantities in the unaccretedMass property of the HotHaloStandard component.

<logical> unaccretedMassIsGettable() Returns true if the unaccretedMass property is gettable for the hotHalo component class.

<logical> unaccretedMassIsSettable() Specify whether the unaccretedMass property of the hotHalo component is settable.

<void> unaccretedMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interrupt [interruptProcedure]↔) Accept a rate set for the unaccretedMass property of the hotHalo component class. Trigger an interrupt to create the component.

<double precision> unaccretedMassRateGet() Returns a zero rate for the unaccretedMass property for the hotHalo component class.

<void> unaccretedMassScale(<double precision> value) Set the scale of the unaccretedMass property of the HotHaloStandard component.

<void> unaccretedMassSet(<double precision> value) Set the unaccretedMass property of the hotHalo component.

<logical> verySimpleDelayedIsActive() Return true if the verySimpleDelayed implementation of the hotHalo component is the active choice.

<logical> verySimpleIsActive() Return true if the verySimple implementation of the hotHalo component is the active choice.

nodeComponentHotHaloColdMode

<type(abundances)> abundances() Get the abundances property of an standard implementation of the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> abundancesAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the abundances property

<type(abundances)> abundancesCold() Get the abundancesCold property of an coldMode implementation of the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> abundancesColdAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the abundancesCold property

<integer> abundancesColdCount() Return a count of the number of scalar properties in the abundancesCold property of an coldMode implementation of the hotHalo component class.

<void> abundancesColdInactive() Indicate that the abundancesCold property of an coldMode implementation of the hotHalo component class is inactive for differential equation solving.

<logical> abundancesColdIsGettable() Returns true if the abundancesCold property is gettable for the hotHalo component class.

<logical> abundancesColdIsSettable() Specify whether the abundancesCold property of the hotHalo component is settable.

<void> abundancesColdRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interrupt [interruptProcedure]↔) Accumulate to the rate of change of the abundancesCold property of an coldMode implementation of the hotHalo component class.

<type(abundances)> abundancesColdRateGet() Get the rate of change of the abundancesCold property of an coldMode implementation of the hotHalo component class.

<void> abundancesColdScale(<type(abundances)> setValue→) Set the absolute scale of the abundancesCold property of an coldMode implementation of the hotHalo component class.

<void> abundancesColdSet(<type(abundances)> setValue→) Set the abundancesCold property of an coldMode implementation of the hotHalo component class.

<integer> abundancesCount() Return a count of the number of scalar properties in the abundances property of an standard implementation of the hotHalo component class.

<void> abundancesInactive() Indicate that the abundances property of an standard implementation of the hotHalo component class is inactive for differential equation solving.

<logical> abundancesIsGettable() Returns true if the abundances property is gettable for the hotHalo component class.

<logical> abundancesIsSettable() Specify whether the abundances property of the hotHalo component is settable.

<void> abundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask [interruptProcedure]↔) Accumulate to the rate of change of the abundances property of an standard implementation of the hotHalo component class.

<type(abundances)> abundancesRateGet() Get the rate of change of the abundances property of an standard implementation of the hotHalo component class.

<void> abundancesScale(<type(abundances)> setValue→) Set the absolute scale of the abundances property of an standard implementation of the hotHalo component class.

<void> abundancesSet(<type(abundances)> setValue→) Set the abundances property of an standard implementation of the hotHalo component class.

<double precision> angularMomentum() Get the angularMomentum property of an standard implementation of the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> angularMomentumAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the angularMomentum property

<double precision> angularMomentumCold() Get the angularMomentumCold property of an coldMode implementation of the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> angularMomentumColdAttributeMatch(<logica [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the angularMomentumCold property

<integer> angularMomentumColdCount() Return a count of the number of scalar properties in the angularMomentumCold property of an coldMode implementation of the hotHalo component class.

<void> angularMomentumColdInactive() Indicate that the angularMomentumCold property of an coldMode implementation of the hotHalo component class is inactive for differential equation solving.

<logical> angularMomentumColdIsGettable() Returns true if the angularMomentumCold property is gettable for the hotHalo component class.

<logical> angularMomentumColdIsSettable() Specify whether the angularMomentumCold property of the hotHalo component is settable.

<void> angularMomentumColdRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the angularMomentumCold property of an coldMode implementation of the hotHalo component class.

<double precision> angularMomentumColdRateGet() Get the rate of change of the angularMomentumCold property of an coldMode implementation of the hotHalo component class.

<void> angularMomentumColdScale(<double precision> setValue→) Set the absolute scale of the angularMomentumCold property of an coldMode implementation of the hotHalo component class.

<void> angularMomentumColdSet(<double precision> setValue→) Set the angularMomentumCold property of an coldMode implementation of the hotHalo component class.

<integer> angularMomentumCount() Return a count of the number of scalar properties in the angularMomentum property of an standard implementation of the hotHalo component class.

<void> angularMomentumInactive() Indicate that the angularMomentum property of an standard implementation of the hotHalo component class is inactive for differential equation solving.

<logical> angularMomentumIsGettable() Returns true if the angularMomentum property is gettable for the hotHalo component class.

<logical> angularMomentumIsSettable() Specify whether the angularMomentum property of the hotHalo component is settable.

<void> angularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interrup [interruptProcedure]↔) Accumulate to the rate of change of the angularMomentum property of an standard implementation of the hotHalo component class.

<double precision> angularMomentumRateGet() Get the rate of change of the angularMomentum property of an standard implementation of the hotHalo component class.

<void> angularMomentumScale(<double precision> setValue→) Set the absolute scale of the angularMomentum property of an standard implementation of the hotHalo component class.

<void> angularMomentumSet(<double precision> setValue→) Set the angularMomentum property of an standard implementation of the hotHalo component class.

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a coldMode implementation of the hotHalo component from a supplied XML definition.

<type(chemicalAbundances)> chemicals() Get the chemicals property of an standard implementation of the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> chemicalsAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the chemicals property

<integer> chemicalsCount() Return a count of the number of scalar properties in the chemicals property of an standard implementation of the hotHalo component class.

<void> chemicalsInactive() Indicate that the chemicals property of an standard implementation of the hotHalo component class is inactive for differential equation solving.

<logical> chemicalsIsGettable() Returns true if the chemicals property is gettable for the hotHalo component class.

<logical> chemicalsIsSettable() Specify whether the chemicals property of the hotHalo component is settable.

<void> chemicalsRate(<type(chemicalAbundances)> setValue→, <logical> [interrupt]↔, <*procedure(interr [interruptProcedure]↔) Accumulate to the rate of change of the chemicals property of an standard implementation of the hotHalo component class.

<type(chemicalAbundances)> chemicalsRateGet() Get the rate of change of the chemicals property of an standard implementation of the hotHalo component class.

<void> chemicalsScale(<type(chemicalAbundances)> setValue→) Set the absolute scale of the chemicals property of an standard implementation of the hotHalo component class.

<void> chemicalsSet(<type(chemicalAbundances)> setValue→) Set the chemicals property of an standard implementation of the hotHalo component class.

<logical> coldModeIsActive() Return true if the coldMode implementation of the hotHalo component is the active choice.

<void> createFunctionSet() Set the create function for the standard implementation of the hotHalo component class.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a coldMode implementation of the hotHalo component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a coldMode implementation of the hotHalo component from array.

<void> destroy() Finalize a coldMode implementation of the hotHalo component.

<void> dumpASCII() Dump the content of a hotHalo component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<double precision> heatSource() Returns the default value for the heatSource property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> heatSourceAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the heatSource property

<integer> heatSourceCount() Compute the count of evolvable quantities in the heatSource property of the HotHaloStandard component.

<logical> heatSourceIsGettable() Returns true if the heatSource property is gettable for the hotHalo component class.

<logical> heatSourceIsSettable() Specify whether the heatSource property of the hotHalo component is settable.

<void> heatSourceRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask [interruptProcedure]↔) Accumulate the rate of change of the heatSource property of the standard implementation of the hotHalo component using a deferred function.

<void> heatSourceRateFunction() Set the function to be used for the rate method of the heatSource property of the HotHaloStandard component.

<double precision> heatSourceRateGet() Returns a zero rate for the heatSource property for the hotHalo component class.

<logical> heatSourceRateIsAttached() Return true if the deferred function used to rate the heatSource property of the HotHaloStandard component class has been attached.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<type(abundances)> hotHaloCoolingAbundances() Returns the default value for the hotHaloCoolingAbundances property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> hotHaloCoolingAbundancesAttributeMatch(<l [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the hotHaloCoolingAbundances property

<integer> hotHaloCoolingAbundancesCount() Compute the count of evolvable quantities in the hotHaloCoolingAbundan property of the HotHaloStandard component.

<logical> hotHaloCoolingAbundancesIsGettable() Returns true if the hotHaloCoolingAbundances property is gettable for the hotHalo component class.

<logical> hotHaloCoolingAbundancesIsSettable() Specify whether the hotHaloCoolingAbundances property of the hotHalo component is settable.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAbundancesRateFunction() Set the function to be used for the rate method of the hotHaloCoolingAbundances property of the hotHalo component.

<type(abundances)> hotHaloCoolingAbundancesRateGet() Returns a zero rate for the hotHaloCoolingAbundances property for the hotHalo component class.

<logical> hotHaloCoolingAbundancesRateIsAttached() Return true if the deferred function used to rate the hotHaloCoolingAbundances property of the hotHalo component class has been attached.

<double precision> hotHaloCoolingAngularMomentum() Returns the default value for the hotHaloCoolingAngularMome property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> hotHaloCoolingAngularMomentumAttributeMat [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the hotHaloCoolingAngularMomentum property

<integer> hotHaloCoolingAngularMomentumCount() Compute the count of evolvable quantities in the hotHaloCoolingAngularMomentum property of the HotHaloStandard component.

<logical> hotHaloCoolingAngularMomentumIsGettable() Returns true if the hotHaloCoolingAngularMomentum property is gettable for the hotHalo component class.

<logical> hotHaloCoolingAngularMomentumIsSettable() Specify whether the hotHaloCoolingAngularMomentum property of the hotHalo component is settable.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRateFunction() Set the function to be used for the rate method of the hotHaloCoolingAngularMomentum property of the hotHalo component.

<double precision> hotHaloCoolingAngularMomentumRateGet() Returns a zero rate for the hotHaloCoolingAngularMo property for the hotHalo component class.

<logical> hotHaloCoolingAngularMomentumRateIsAttached() Return true if the deferred function used to rate the hotHaloCoolingAngularMomentum property of the hotHalo component class has been attached.

<double precision> hotHaloCoolingMass() Returns the default value for the hotHaloCoolingMass property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> hotHaloCoolingMassAttributeMatch(<logical [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the hotHaloCoolingMass property

<integer> hotHaloCoolingMassCount() Compute the count of evolvable quantities in the hotHaloCoolingMass property of the HotHaloStandard component.

<logical> hotHaloCoolingMassIsGettable() Returns true if the hotHaloCoolingMass property is gettable for the hotHalo component class.

<logical> hotHaloCoolingMassIsSettable() Specify whether the hotHaloCoolingMass property of the hotHalo component is settable.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRateFunction() Set the function to be used for the rate method of the hotHaloCoolingMass property of the hotHalo component.

<double precision> hotHaloCoolingMassRateGet() Returns a zero rate for the hotHaloCoolingMass property for the hotHalo component class.

<logical> hotHaloCoolingMassRateIsAttached() Return true if the deferred function used to rate the hotHaloCoolingMass property of the hotHalo component class has been attached.

<void> initialize() Initialize a coldMode member of the hotHalo component.

<logical> isInitialized() Get the isInitialized property of an standard implementation of the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> isInitializedAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the isInitialized property

<logical> isInitializedIsGettable() Returns true if the isInitialized property is gettable for the hotHalo component class.

<logical> isInitializedIsSettable() Specify whether the isInitialized property of the hotHalo component is settable.

<logical> isInitializedRateGet() Returns a zero rate for the isInitialized property for the hotHalo component class.

<void> isInitializedSet(<logical> setValue→) Set the isInitialized property of an standard implementation of the hotHalo component class.

<double precision> mass() Get the mass property of an standard implementation of the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> massAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the mass property

<double precision> massCold() Get the massCold property of an coldMode implementation of the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> massColdAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the massCold property

<integer> massColdCount() Return a count of the number of scalar properties in the massCold property of an coldMode implementation of the hotHalo component class.

<void> massColdInactive() Indicate that the massCold property of an coldMode implementation of the hotHalo component class is inactive for differential equation solving.

<logical> massColdIsGettable() Returns true if the massCold property is gettable for the hotHalo component class.

<logical> massColdIsSettable() Specify whether the massCold property of the hotHalo component is settable.

<void> massColdRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the massCold property of an coldMode implementation of the hotHalo component class.

<double precision> massColdRateGet() Get the rate of change of the massCold property of an coldMode implementation of the hotHalo component class.

<void> massColdScale(<double precision> setValue→) Set the absolute scale of the massCold property of an coldMode implementation of the hotHalo component class.

<void> massColdSet(<double precision> setValue→) Set the massCold property of an coldMode implementation of the hotHalo component class.

<integer> massCount() Return a count of the number of scalar properties in the mass property of an standard implementation of the hotHalo component class.

<void> massInactive() Indicate that the mass property of an standard implementation of the hotHalo component class is inactive for differential equation solving.

<logical> massIsGettable() Returns true if the mass property is gettable for the hotHalo component class.

<logical> massIsSettable() Specify whether the mass property of the hotHalo component is settable.

<void> massRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the mass property of an standard implementation of the hotHalo component class.

<double precision> massRateGet() Get the rate of change of the mass property of an standard implementation of the hotHalo component class.

<void> massRemovalRate() Called whenever the standard hot halo component removes mass from the halo.

<void> massScale(<double precision> setValue→) Set the absolute scale of the mass property of an standard implementation of the hotHalo component class.

<void> massSet(<double precision> setValue→) Set the mass property of an standard implementation of the hotHalo component class.

<double precision> massSink() Returns the default value for the massSink property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> massSinkAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the massSink property

<integer> massSinkCount() Compute the count of evolvable quantities in the massSink property of the HotHaloStandard component.

<logical> massSinkIsGettable() Returns true if the massSink property is gettable for the hotHalo component class.

<logical> massSinkIsSettable() Specify whether the massSink property of the hotHalo component is settable.

<void> massSinkRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the massSink property of the standard implementation of the hotHalo component using a deferred function.

<void> massSinkRateFunction() Set the function to be used for the rate method of the massSink property of the HotHaloStandard component.

<double precision> massSinkRateGet() Returns a zero rate for the massSink property for the hotHalo component class.

<logical> massSinkRateIsAttached() Return true if the deferred function used to rate the massSink property of the HotHaloStandard component class has been attached.

<double precision> massTotal() Returns the default value for the massTotal property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> massTotalAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the massTotal property

<logical> massTotalIsGettable() Returns true if the massTotal property is gettable for the hotHalo component class.

<logical> massTotalIsSettable() Specify whether the massTotal property of the hotHalo component is settable.

<double precision> massTotalRateGet() Returns a zero rate for the massTotal property for the hotHalo component class.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a coldMode implementation of the hotHalo component class.

<logical> nullIsActive() Return true if the null implementation of the hotHalo component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<double precision> outerRadius() Get the value of the outerRadius property of the standard implementation of the hotHalo component using a deferred function.

<type(varying_string), allocatable, dimension(:) => matches> outerRadiusAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the outerRadius property

<integer> outerRadiusCount() Return a count of the number of scalar properties in the outerRadius property of an standard implementation of the hotHalo component class.

<void> outerRadiusFunction() Set the function to be used for the get method of the outerRadius property of the HotHaloStandard component.

<double precision> outerRadiusGrowthRate() Call the deferred function for the outerRadiusGrowthRate method of the hotHalo component class if it has been set.

<void> outerRadiusGrowthRateFunction() Set the function to be used for the outerRadiusGrowthRate method of the coldMode implementation of the hotHalo component class.

<logical> outerRadiusGrowthRateFunctionIsSet() Return true if the deferred function for the outerRadiusGrowthRate method of the coldMode implementation of the hotHalo component class has been set.

<void> outerRadiusInactive() Indicate that the outerRadius property of an standard implementation of the hotHalo component class is inactive for differential equation solving.

<logical> outerRadiusIsAttached() Return true if the deferred function used to get the outerRadius property of the HotHaloStandard component class has been attached.

<logical> outerRadiusIsGettable() Returns true if the outerRadius property is gettable for the hotHalo component class.

<logical> outerRadiusIsSettable() Specify whether the outerRadius property of the hotHalo component is settable.

<void> outerRadiusRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTas [interruptProcedure]↔) Accumulate to the rate of change of the outerRadius property of an standard implementation of the hotHalo component class.

<double precision> outerRadiusRateGet() Get the rate of change of the outerRadius property of an standard implementation of the hotHalo component class.

<void> outerRadiusScale(<double precision> setValue→) Set the absolute scale of the outerRadius property of an standard implementation of the hotHalo component class.

<void> outerRadiusSet(<double precision> setValue→) Set the outerRadius property of an standard implementation of the hotHalo component class.

<double precision> outerRadiusValue() Get the outerRadius property of an standard implementation of the hotHalo component class.

<type(abundances)> outflowedAbundances() Get the outflowedAbundances property of an standard implementation of the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> outflowedAbundancesAttributeMatch(<logica [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the outflowedAbundances property

<integer> outflowedAbundancesCount() Return a count of the number of scalar properties in the outflowedAbundances property of an standard implementation of the hotHalo component class.

<void> outflowedAbundancesInactive() Indicate that the outflowedAbundances property of an standard implementation of the hotHalo component class is inactive for differential equation solving.

<logical> outflowedAbundancesIsGettable() Returns true if the outflowedAbundances property is gettable for the hotHalo component class.

<logical> outflowedAbundancesIsSettable() Specify whether the outflowedAbundances property of the hotHalo component is settable.

<void> outflowedAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the outflowedAbundances property of an standard implementation of the hotHalo component class.

<type(abundances)> outflowedAbundancesRateGet() Get the rate of change of the outflowedAbundances property of an standard implementation of the hotHalo component class.

<void> outflowedAbundancesScale(<type(abundances)> setValue→) Set the absolute scale of the outflowedAbundances property of an standard implementation of the hotHalo component class.

<void> outflowedAbundancesSet(<type(abundances)> setValue→) Set the outflowedAbundances property of an standard implementation of the hotHalo component class.

<double precision> outflowedAngularMomentum() Get the outflowedAngularMomentum property of an standard implementation of the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> outflowedAngularMomentumAttributeMatch(<l [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the outflowedAngularMomentum property

<integer> outflowedAngularMomentumCount() Return a count of the number of scalar properties in the outflowedAngularMomentum property of an standard implementation of the hotHalo component class.

<void> outflowedAngularMomentumInactive() Indicate that the outflowedAngularMomentum property of an standard implementation of the hotHalo component class is inactive for differential equation solving.

<logical> outflowedAngularMomentumIsGettable() Returns true if the outflowedAngularMomentum property is gettable for the hotHalo component class.

<logical> outflowedAngularMomentumIsSettable() Specify whether the outflowedAngularMomentum property of the hotHalo component is settable.

<void> outflowedAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the outflowedAngularMomentum property of an standard implementation of the hotHalo component class.

<double precision> outflowedAngularMomentumRateGet() Get the rate of change of the outflowedAngularMomentum property of an standard implementation of the hotHalo component class.

<void> outflowedAngularMomentumScale(<double precision> setValue→) Set the absolute scale of the outflowedAngularMomentum property of an standard implementation of the hotHalo component class.

<void> outflowedAngularMomentumSet(<double precision> setValue→) Set the outflowedAngularMomentum property of an standard implementation of the hotHalo component class.

<double precision> outflowedMass() Get the outflowedMass property of an standard implementation of the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> outflowedMassAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the outflowedMass property

<integer> outflowedMassCount() Return a count of the number of scalar properties in the outflowedMass property of an standard implementation of the hotHalo component class.

<void> outflowedMassInactive() Indicate that the outflowedMass property of an standard implementation of the hotHalo component class is inactive for differential equation solving.

<logical> outflowedMassIsGettable() Returns true if the outflowedMass property is gettable for the hotHalo component class.

<logical> outflowedMassIsSettable() Specify whether the outflowedMass property of the hotHalo component is settable.

<void> outflowedMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptT [interruptProcedure]↔) Accumulate to the rate of change of the outflowedMass property of an standard implementation of the hotHalo component class.

<double precision> outflowedMassRateGet() Get the rate of change of the outflowedMass property of an standard implementation of the hotHalo component class.

<void> outflowedMassScale(<double precision> setValue→) Set the absolute scale of the outflowedMass property of an standard implementation of the hotHalo component class.

<void> outflowedMassSet(<double precision> setValue→) Set the outflowedMass property of an standard implementation of the hotHalo component class.

<type(abundances)> outflowingAbundances() Returns the default value for the outflowingAbundances property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> outflowingAbundancesAttributeMatch(<logic [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the outflowingAbundances property

<integer> outflowingAbundancesCount() Compute the count of evolvable quantities in the outflowingAbundances property of the HotHaloStandard component.

<logical> outflowingAbundancesIsGettable() Returns true if the outflowingAbundances property is gettable for the hotHalo component class.

<logical> outflowingAbundancesIsSettable() Specify whether the outflowingAbundances property of the hotHalo component is settable.

<void> outflowingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the outflowingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> outflowingAbundancesRateFunction() Set the function to be used for the rate method of the outflowingAbundances property of the HotHaloStandard component.

<type(abundances)> outflowingAbundancesRateGet() Returns a zero rate for the outflowingAbundances property for the hotHalo component class.

<logical> outflowingAbundancesRateIsAttached() Return true if the deferred function used to rate the outflowingAbundances property of the HotHaloStandard component class has been attached.

<double precision> outflowingAngularMomentum() Returns the default value for the outflowingAngularMomentum property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> outflowingAngularMomentumAttributeMatch(< [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the outflowingAngularMomentum property

<integer> outflowingAngularMomentumCount() Compute the count of evolvable quantities in the outflowingAngularMomentum property of the HotHaloStandard component.

<logical> outflowingAngularMomentumIsGettable() Returns true if the outflowingAngularMomentum property is gettable for the hotHalo component class.

<logical> outflowingAngularMomentumIsSettable() Specify whether the outflowingAngularMomentum property of the hotHalo component is settable.

<void> outflowingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the outflowingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> outflowingAngularMomentumRateFunction() Set the function to be used for the rate method of the outflowingAngularMomentum property of the HotHaloStandard component.

<double precision> outflowingAngularMomentumRateGet() Returns a zero rate for the outflowingAngularMomentum property for the hotHalo component class.

<logical> outflowingAngularMomentumRateIsAttached() Return true if the deferred function used to rate the outflowingAngularMomentum property of the HotHaloStandard component class has been attached.

<double precision> outflowingMass() Returns the default value for the outflowingMass property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> outflowingMassAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the outflowingMass property

<integer> outflowingMassCount() Compute the count of evolvable quantities in the outflowingMass property of the HotHaloStandard component.

<logical> outflowingMassIsGettable() Returns true if the outflowingMass property is gettable for the hotHalo component class.

<logical> outflowingMassIsSettable() Specify whether the outflowingMass property of the hotHalo component is settable.

<void> outflowingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interrupt [interruptProcedure]↔) Accumulate the rate of change of the outflowingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> outflowingMassRateFunction() Set the function to be used for the rate method of the outflowingMass property of the HotHaloStandard component.

<double precision> outflowingMassRateGet() Returns a zero rate for the outflowingMass property for the hotHalo component class.

<logical> outflowingMassRateIsAttached() Return true if the deferred function used to rate the outflowingMass property of the HotHaloStandard component class has been attached.

<void> outflowReturn(<logical> interrupt↔, <*procedure(interruptTask)> interruptprocedure↔) Call the deferred function for the outflowReturn method of the hotHalo component class if it has been set.

<void> outflowReturnFunction() Set the function to be used for the outflowReturn method of the coldMode implementation of the hotHalo component class.

<logical> outflowReturnFunctionIsSet() Return true if the deferred function for the outflowReturn method of the coldMode implementation of the hotHalo component class has been set.

<logical> outflowTrackingIsActive() Return true if the outflowTracking implementation of the hotHalo component is the active choice.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a hotHalo component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a coldMode implementation of the hotHalo component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a coldMode implementation of the hotHalo component.

<void> postOutput(<double precision> time→) Perform post-output processing for a coldMode implementation of the hotHalo component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a coldMode implementation of the hotHalo component.

<void> serializeASCII() Serialize the contents of a coldMode implementation of the hotHalo component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a coldMode implementation of the hotHalo component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a coldMode implementation of the hotHalo component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a coldMode implementation of the hotHalo component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a coldMode implementation of the hotHalo component to XML.

<integer(c_size_t)> sizeOf() Return the size in bytes of a coldMode implementation of the hotHalo component.

<logical> standardIsActive() Return true if the standard implementation of the hotHalo component is the active choice.

<type(abundances)> strippedAbundances() Get the strippedAbundances property of an standard implementation of the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> strippedAbundancesAttributeMatch(<logical [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the strippedAbundances property

<integer> strippedAbundancesCount() Return a count of the number of scalar properties in the strippedAbundances property of an standard implementation of the hotHalo component class.

<void> strippedAbundancesInactive() Indicate that the strippedAbundances property of an standard implementation of the hotHalo component class is inactive for differential equation solving.

<logical> strippedAbundancesIsGettable() Returns true if the strippedAbundances property is gettable for the hotHalo component class.

<logical> strippedAbundancesIsSettable() Specify whether the strippedAbundances property of the hotHalo component is settable.

<void> strippedAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate to the rate of change of the strippedAbundances property of an standard implementation of the hotHalo component class.

<type(abundances)> strippedAbundancesRateGet() Get the rate of change of the strippedAbundances property of an standard implementation of the hotHalo component class.

<void> strippedAbundancesScale(<type(abundances)> setValue→) Set the absolute scale of the strippedAbundances property of an standard implementation of the hotHalo component class.

<void> strippedAbundancesSet(<type(abundances)> setValue→) Set the strippedAbundances property of an standard implementation of the hotHalo component class.

<double precision> strippedMass() Get the strippedMass property of an standard implementation of the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> strippedMassAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the strippedMass property

<integer> strippedMassCount() Return a count of the number of scalar properties in the strippedMass property of an standard implementation of the hotHalo component class.

<void> strippedMassInactive() Indicate that the strippedMass property of an standard implementation of the hotHalo component class is inactive for differential equation solving.

<logical> strippedMassIsGettable() Returns true if the strippedMass property is gettable for the hotHalo component class.

<logical> strippedMassIsSettable() Specify whether the strippedMass property of the hotHalo component is settable.

<void> strippedMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTa [interruptProcedure]↔) Accumulate to the rate of change of the strippedMass property of an standard implementation of the hotHalo component class.

<double precision> strippedMassRateGet() Get the rate of change of the strippedMass property of an standard implementation of the hotHalo component class.

<void> strippedMassScale(<double precision> setValue→) Set the absolute scale of the strippedMass property of an standard implementation of the hotHalo component class.

<void> strippedMassSet(<double precision> setValue→) Set the strippedMass property of an standard implementation of the hotHalo component class.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<type(abundances)> trackedOutflowAbundances() Returns the default value for the trackedOutflowAbundances property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> trackedOutflowAbundancesAttributeMatch(<l [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the trackedOutflowAbundances property

<integer> trackedOutflowAbundancesCount() Compute the count of evolvable quantities in the trackedOutflowAbundan property of the HotHaloOutflowTracking component.

<logical> trackedOutflowAbundancesIsGettable() Returns true if the trackedOutflowAbundances property is gettable for the hotHalo component class.

<logical> trackedOutflowAbundancesIsSettable() Specify whether the trackedOutflowAbundances property of the hotHalo component is settable.

<void> trackedOutflowAbundancesRate(<type(abundances)> value) Cumulate to the rate of the trackedOutflowAbundances property of the HotHaloOutflowTracking component.

<type(abundances)> trackedOutflowAbundancesRateGet() Returns a zero rate for the trackedOutflowAbundances property for the hotHalo component class.

<void> trackedOutflowAbundancesScale(<type(abundances)> value) Set the scale of the trackedOutflowAbundances property of the HotHaloOutflowTracking component.

<void> trackedOutflowAbundancesSet(<type(abundances)> value) Set the trackedOutflowAbundances property of the hotHalo component.

<double precision> trackedOutflowMass() Returns the default value for the trackedOutflowMass property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> trackedOutflowMassAttributeMatch(<logical [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the trackedOutflowMass property

<integer> trackedOutflowMassCount() Compute the count of evolvable quantities in the trackedOutflowMass property of the HotHaloOutflowTracking component.

<logical> trackedOutflowMassIsGettable() Returns true if the trackedOutflowMass property is gettable for the hotHalo component class.

<logical> trackedOutflowMassIsSettable() Specify whether the trackedOutflowMass property of the hotHalo component is settable.

<void> trackedOutflowMassRate(<double precision> value) Cumulate to the rate of the trackedOutflowMass property of the HotHaloOutflowTracking component.

<double precision> trackedOutflowMassRateGet() Returns a zero rate for the trackedOutflowMass property for the hotHalo component class.

<void> trackedOutflowMassScale(<double precision> value) Set the scale of the trackedOutflowMass property of the HotHaloOutflowTracking component.

<void> trackedOutflowMassSet(<double precision> value) Set the trackedOutflowMass property of the hotHalo component.

<type(varying_string)> type() Returns the type name for the coldMode implementation of the hotHalo component class.

<double precision> unaccretedMass() Get the unaccretedMass property of an standard implementation of the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> unaccretedMassAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the unaccretedMass property

<integer> unaccretedMassCount() Return a count of the number of scalar properties in the unaccretedMass property of an standard implementation of the hotHalo component class.

<void> unaccretedMassInactive() Indicate that the unaccretedMass property of an standard implementation of the hotHalo component class is inactive for differential equation solving.

<logical> unaccretedMassIsGettable() Returns true if the unaccretedMass property is gettable for the hotHalo component class.

<logical> unaccretedMassIsSettable() Specify whether the unaccretedMass property of the hotHalo component is settable.

<void> unaccretedMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interrupt [interruptProcedure]↔) Accumulate to the rate of change of the unaccretedMass property of an standard implementation of the hotHalo component class.

<double precision> unaccretedMassRateGet() Get the rate of change of the unaccretedMass property of an standard implementation of the hotHalo component class.

<void> unaccretedMassScale(<double precision> setValue→) Set the absolute scale of the unaccretedMass property of an standard implementation of the hotHalo component class.

<void> unaccretedMassSet(<double precision> setValue→) Set the unaccretedMass property of an standard implementation of the hotHalo component class.

<logical> verySimpleDelayedIsActive() Return true if the verySimpleDelayed implementation of the hotHalo component is the active choice.

<logical> verySimpleIsActive() Return true if the verySimple implementation of the hotHalo component is the active choice.

nodeComponentHotHaloNull

<type(abundances)> abundances() Returns the default value for the abundances property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> abundancesAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the abundances property

<type(abundances)> abundancesCold() Returns the default value for the abundancesCold property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> abundancesColdAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the abundancesCold property

<integer> abundancesColdCount() Compute the count of evolvable quantities in the abundancesCold property of the HotHaloColdMode component.

<logical> abundancesColdIsGettable() Returns true if the abundancesCold property is gettable for the hotHalo component class.

<logical> abundancesColdIsSettable() Specify whether the abundancesCold property of the hotHalo component is settable.

<void> abundancesColdRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interrupt [interruptProcedure]↔) Accept a rate set for the abundancesCold property of the hotHalo component class. Trigger an interrupt to create the component.

<type(abundances)> abundancesColdRateGet() Returns a zero rate for the abundancesCold property for the hotHalo component class.

<void> abundancesColdScale(<type(abundances)> value) Set the scale of the abundancesCold property of the HotHaloColdMode component.

<void> abundancesColdSet(<type(abundances)> value) Set the abundancesCold property of the hotHalo component.

<integer> abundancesCount() Compute the count of evolvable quantities in the abundances property of the HotHaloStandard component.

<logical> abundancesIsGettable() Returns true if the abundances property is gettable for the hotHalo component class.

<logical> abundancesIsSettable() Specify whether the abundances property of the hotHalo component is settable.

<void> abundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask [interruptProcedure]↔) Accept a rate set for the abundances property of the hotHalo component class. Trigger an interrupt to create the component.

<type(abundances)> abundancesRateGet() Returns a zero rate for the abundances property for the hotHalo component class.

<void> abundancesScale(<type(abundances)> value) Set the scale of the abundances property of the HotHaloStandard component.

<void> abundancesSet(<type(abundances)> value) Set the abundances property of the hotHalo component.

<double precision> angularMomentum() Returns the default value for the angularMomentum property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> angularMomentumAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the angularMomentum property

<double precision> angularMomentumCold() Returns the default value for the angularMomentumCold property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> angularMomentumColdAttributeMatch(<logica [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the angularMomentumCold property

<integer> angularMomentumColdCount() Compute the count of evolvable quantities in the angularMomentumCold property of the HotHaloColdMode component.

<logical> angularMomentumColdIsGettable() Returns true if the angularMomentumCold property is gettable for the hotHalo component class.

<logical> angularMomentumColdIsSettable() Specify whether the angularMomentumCold property of the hotHalo component is settable.

<void> angularMomentumColdRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accept a rate set for the angularMomentumCold property of the hotHalo component class. Trigger an interrupt to create the component.

<double precision> angularMomentumColdRateGet() Returns a zero rate for the angularMomentumCold property for the hotHalo component class.

<void> angularMomentumColdScale(<double precision> value) Set the scale of the angularMomentumCold property of the HotHaloColdMode component.

<void> angularMomentumColdSet(<double precision> value) Set the angularMomentumCold property of the hotHalo component.

<integer> angularMomentumCount() Compute the count of evolvable quantities in the angularMomentum property of the HotHaloStandard component.

<logical> angularMomentumIsGettable() Returns true if the angularMomentum property is gettable for the hotHalo component class.

<logical> angularMomentumIsSettable() Specify whether the angularMomentum property of the hotHalo component is settable.

<void> angularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interrup [interruptProcedure]↔) Accept a rate set for the angularMomentum property of the hotHalo component class. Trigger an interrupt to create the component.

<double precision> angularMomentumRateGet() Returns a zero rate for the angularMomentum property for the hotHalo component class.

<void> angularMomentumScale(<double precision> value) Set the scale of the angularMomentum property of the HotHaloStandard component.

<void> angularMomentumSet(<double precision> value) Set the angularMomentum property of the hotHalo component.

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a null implementation of the hotHalo component from a supplied XML definition.

<type(chemicalAbundances)> chemicals() Returns the default value for the chemicals property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> chemicalsAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the chemicals property

<integer> chemicalsCount() Compute the count of evolvable quantities in the chemicals property of the HotHaloStandard component.

<logical> chemicalsIsGettable() Returns true if the chemicals property is gettable for the hotHalo component class.

<logical> chemicalsIsSettable() Specify whether the chemicals property of the hotHalo component is settable.

<void> chemicalsRate(<type(chemicalAbundances)> setValue→, <logical> [interrupt]↔, <*procedure(interr [interruptProcedure]↔) Accept a rate set for the chemicals property of the hotHalo component class. Trigger an interrupt to create the component.

<type(chemicalAbundances)> chemicalsRateGet() Returns a zero rate for the chemicals property for the hotHalo component class.

<void> chemicalsScale(<type(chemicalAbundances)> value) Set the scale of the chemicals property of the HotHaloStandard component.

<void> chemicalsSet(<type(chemicalAbundances)> value) Set the chemicals property of the hotHalo component.

<logical> coldModeIsActive() Return true if the coldMode implementation of the hotHalo component is the active choice.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a null implementation of the hotHalo component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a null implementation of the hotHalo component from array.

<void> destroy() Finalize a null implementation of the hotHalo component.

<void> dumpASCII() Dump the content of a hotHalo component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<double precision> heatSource() Returns the default value for the heatSource property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> heatSourceAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the heatSource property

<integer> heatSourceCount() Compute the count of evolvable quantities in the heatSource property of the HotHaloStandard component.

<logical> heatSourceIsGettable() Returns true if the heatSource property is gettable for the hotHalo component class.

<logical> heatSourceIsSettable() Specify whether the heatSource property of the hotHalo component is settable.

<void> heatSourceRate(<double precision> value) Cumulate to the rate of the heatSource property of the HotHaloStandard component.

<double precision> heatSourceRateGet() Returns a zero rate for the heatSource property for the hotHalo component class.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<type(abundances)> hotHaloCoolingAbundances() Returns the default value for the hotHaloCoolingAbundances property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> hotHaloCoolingAbundancesAttributeMatch(<l [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the hotHaloCoolingAbundances property

<integer> hotHaloCoolingAbundancesCount() Compute the count of evolvable quantities in the hotHaloCoolingAbundan property of the HotHaloStandard component.

<logical> hotHaloCoolingAbundancesIsGettable() Returns true if the hotHaloCoolingAbundances property is gettable for the hotHalo component class.

<logical> hotHaloCoolingAbundancesIsSettable() Specify whether the hotHaloCoolingAbundances property of the hotHalo component is settable.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAbundancesRateFunction() Set the function to be used for the rate method of the hotHaloCoolingAbundances property of the hotHalo component.

<type(abundances)> hotHaloCoolingAbundancesRateGet() Returns a zero rate for the hotHaloCoolingAbundances property for the hotHalo component class.

<logical> hotHaloCoolingAbundancesRateIsAttached() Return true if the deferred function used to rate the hotHaloCoolingAbundances property of the hotHalo component class has been attached.

<double precision> hotHaloCoolingAngularMomentum() Returns the default value for the hotHaloCoolingAngularMome property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> hotHaloCoolingAngularMomentumAttributeMat [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the hotHaloCoolingAngularMomentum property

<integer> hotHaloCoolingAngularMomentumCount() Compute the count of evolvable quantities in the hotHaloCoolingAngularMomentum property of the HotHaloStandard component.

<logical> hotHaloCoolingAngularMomentumIsGettable() Returns true if the hotHaloCoolingAngularMomentum property is gettable for the hotHalo component class.

<logical> hotHaloCoolingAngularMomentumIsSettable() Specify whether the hotHaloCoolingAngularMomentum property of the hotHalo component is settable.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRateFunction() Set the function to be used for the rate method of the hotHaloCoolingAngularMomentum property of the hotHalo component.

<double precision> hotHaloCoolingAngularMomentumRateGet() Returns a zero rate for the hotHaloCoolingAngularMo property for the hotHalo component class.

<logical> hotHaloCoolingAngularMomentumRateIsAttached() Return true if the deferred function used to rate the hotHaloCoolingAngularMomentum property of the hotHalo component class has been attached.

<double precision> hotHaloCoolingMass() Returns the default value for the hotHaloCoolingMass property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> hotHaloCoolingMassAttributeMatch(<logical [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the hotHaloCoolingMass property

<integer> hotHaloCoolingMassCount() Compute the count of evolvable quantities in the hotHaloCoolingMass property of the HotHaloStandard component.

<logical> hotHaloCoolingMassIsGettable() Returns true if the hotHaloCoolingMass property is gettable for the hotHalo component class.

<logical> hotHaloCoolingMassIsSettable() Specify whether the hotHaloCoolingMass property of the hotHalo component is settable.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRateFunction() Set the function to be used for the rate method of the hotHaloCoolingMass property of the hotHalo component.

<double precision> hotHaloCoolingMassRateGet() Returns a zero rate for the hotHaloCoolingMass property for the hotHalo component class.

<logical> hotHaloCoolingMassRateIsAttached() Return true if the deferred function used to rate the hotHaloCoolingMass property of the hotHalo component class has been attached.

<void> initialize() Initialize a null member of the hotHalo component.

<logical> isInitialized() Returns the default value for the isInitialized property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> isInitializedAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the isInitialized property

<logical> isInitializedIsGettable() Returns true if the isInitialized property is gettable for the hotHalo component class.

<logical> isInitializedIsSettable() Specify whether the isInitialized property of the hotHalo component is settable.

<logical> isInitializedRateGet() Returns a zero rate for the isInitialized property for the hotHalo component class.

<void> isInitializedSet(<logical> value) Set the isInitialized property of the hotHalo component.

<double precision> mass() Returns the default value for the mass property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> massAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the mass property

<double precision> massCold() Returns the default value for the massCold property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> massColdAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the massCold property

<integer> massColdCount() Compute the count of evolvable quantities in the massCold property of the HotHaloColdMode component.

<logical> massColdIsGettable() Returns true if the massCold property is gettable for the hotHalo component class.

<logical> massColdIsSettable() Specify whether the massCold property of the hotHalo component is settable.

<void> massColdRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accept a rate set for the massCold property of the hotHalo component class. Trigger an interrupt to create the component.

<double precision> massColdRateGet() Returns a zero rate for the massCold property for the hotHalo component class.

<void> massColdScale(<double precision> value) Set the scale of the massCold property of the HotHaloColdMode component.

<void> massColdSet(<double precision> value) Set the massCold property of the hotHalo component.

<integer> massCount() Compute the count of evolvable quantities in the mass property of the HotHaloStandard component.

<logical> massIsGettable() Returns true if the mass property is gettable for the hotHalo component class.

<logical> massIsSettable() Specify whether the mass property of the hotHalo component is settable.

<void> massRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accept a rate set for the mass property of the hotHalo component class. Trigger an interrupt to create the component.

<double precision> massRateGet() Returns a zero rate for the mass property for the hotHalo component class.

<void> massScale(<double precision> value) Set the scale of the mass property of the HotHaloStandard component.

<void> massSet(<double precision> value) Set the mass property of the hotHalo component.

<double precision> massSink() Returns the default value for the massSink property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> massSinkAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the massSink property

<integer> massSinkCount() Compute the count of evolvable quantities in the massSink property of the HotHaloStandard component.

<logical> massSinkIsGettable() Returns true if the massSink property is gettable for the hotHalo component class.

<logical> massSinkIsSettable() Specify whether the massSink property of the hotHalo component is settable.

<void> massSinkRate(<double precision> value) Cumulate to the rate of the massSink property of the HotHaloStandard component.

<double precision> massSinkRateGet() Returns a zero rate for the massSink property for the hotHalo component class.

<double precision> massTotal() Returns the default value for the massTotal property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> massTotalAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the massTotal property

<logical> massTotalIsGettable() Returns true if the massTotal property is gettable for the hotHalo component class.

<logical> massTotalIsSettable() Specify whether the massTotal property of the hotHalo component is settable.

<double precision> massTotalRateGet() Returns a zero rate for the massTotal property for the hotHalo component class.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a null implementation of the hotHalo component class.

<logical> nullIsActive() Return true if the null implementation of the hotHalo component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<double precision> outerRadius() Returns the default value for the outerRadius property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> outerRadiusAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the outerRadius property

<integer> outerRadiusCount() Compute the count of evolvable quantities in the outerRadius property of the HotHaloStandard component.

<logical> outerRadiusIsGettable() Returns true if the outerRadius property is gettable for the hotHalo component class.

<logical> outerRadiusIsSettable() Specify whether the outerRadius property of the hotHalo component is settable.

<void> outerRadiusRate(<double precision> value) Cumulate to the rate of the outerRadius property of the HotHaloStandard component.

<double precision> outerRadiusRateGet() Returns a zero rate for the outerRadius property for the hotHalo component class.

<void> outerRadiusScale(<double precision> value) Set the scale of the outerRadius property of the HotHaloStandard component.

<void> outerRadiusSet(<double precision> value) Set the outerRadius property of the hotHalo component.

<type(abundances)> outflowedAbundances() Returns the default value for the outflowedAbundances property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> outflowedAbundancesAttributeMatch(<logica [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the outflowedAbundances property

<integer> outflowedAbundancesCount() Compute the count of evolvable quantities in the outflowedAbundances property of the HotHaloStandard component.

<logical> outflowedAbundancesIsGettable() Returns true if the outflowedAbundances property is gettable for the hotHalo component class.

<logical> outflowedAbundancesIsSettable() Specify whether the outflowedAbundances property of the hotHalo component is settable.

<void> outflowedAbundancesRate(<type(abundances)> value) Cumulate to the rate of the outflowedAbundances property of the HotHaloStandard component.

<type(abundances)> outflowedAbundancesRateGet() Returns a zero rate for the outflowedAbundances property for the hotHalo component class.

<void> outflowedAbundancesScale(<type(abundances)> value) Set the scale of the outflowedAbundances property of the HotHaloStandard component.

<void> outflowedAbundancesSet(<type(abundances)> value) Set the outflowedAbundances property of the hotHalo component.

<double precision> outflowedAngularMomentum() Returns the default value for the outflowedAngularMomentum property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> outflowedAngularMomentumAttributeMatch(<l [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the outflowedAngularMomentum property

<integer> outflowedAngularMomentumCount() Compute the count of evolvable quantities in the outflowedAngularMomen property of the HotHaloStandard component.

<logical> outflowedAngularMomentumIsGettable() Returns true if the outflowedAngularMomentum property is gettable for the hotHalo component class.

<logical> outflowedAngularMomentumIsSettable() Specify whether the outflowedAngularMomentum property of the hotHalo component is settable.

<void> outflowedAngularMomentumRate(<double precision> value) Cumulate to the rate of the outflowedAngularMomentum property of the HotHaloStandard component.

<double precision> outflowedAngularMomentumRateGet() Returns a zero rate for the outflowedAngularMomentum property for the hotHalo component class.

<void> outflowedAngularMomentumScale(<double precision> value) Set the scale of the outflowedAngularMomentum property of the HotHaloStandard component.

<void> outflowedAngularMomentumSet(<double precision> value) Set the outflowedAngularMomentum property of the hotHalo component.

<double precision> outflowedMass() Returns the default value for the outflowedMass property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> outflowedMassAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the outflowedMass property

<integer> outflowedMassCount() Compute the count of evolvable quantities in the outflowedMass property of the HotHaloStandard component.

<logical> outflowedMassIsGettable() Returns true if the outflowedMass property is gettable for the hotHalo component class.

<logical> outflowedMassIsSettable() Specify whether the outflowedMass property of the hotHalo component is settable.

<void> outflowedMassRate(<double precision> value) Cumulate to the rate of the outflowedMass property of the HotHaloStandard component.

<double precision> outflowedMassRateGet() Returns a zero rate for the outflowedMass property for the hotHalo component class.

<void> outflowedMassScale(<double precision> value) Set the scale of the outflowedMass property of the HotHaloStandard component.

<void> outflowedMassSet(<double precision> value) Set the outflowedMass property of the hotHalo component.

<type(abundances)> outflowingAbundances() Returns the default value for the outflowingAbundances property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> outflowingAbundancesAttributeMatch(<logi
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the outflowingAbundances property

<integer> outflowingAbundancesCount() Compute the count of evolvable quantities in the outflowingAbundances property of the HotHaloStandard component.

<logical> outflowingAbundancesIsGettable() Returns true if the outflowingAbundances property is gettable for the hotHalo component class.

<logical> outflowingAbundancesIsSettable() Specify whether the outflowingAbundances property of the hotHalo component is settable.

<void> outflowingAbundancesRate(<type(abundances)> value) Cumulate to the rate of the outflowingAbundances property of the HotHaloStandard component.

<type(abundances)> outflowingAbundancesRateGet() Returns a zero rate for the outflowingAbundances property for the hotHalo component class.

<double precision> outflowingAngularMomentum() Returns the default value for the outflowingAngularMomentum property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> outflowingAngularMomentumAttributeMatch(<
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the hotHalo class that have the desired attributes for the outflowingAngularMomentum property

<integer> outflowingAngularMomentumCount() Compute the count of evolvable quantities in the outflowingAngularMomentum property of the HotHaloStandard component.

<logical> outflowingAngularMomentumIsGettable() Returns true if the outflowingAngularMomentum property is gettable for the hotHalo component class.

<logical> outflowingAngularMomentumIsSettable() Specify whether the outflowingAngularMomentum property of the hotHalo component is settable.

<void> outflowingAngularMomentumRate(<double precision> value) Cumulate to the rate of the outflowingAngularMomentum property of the HotHaloStandard component.

<double precision> outflowingAngularMomentumRateGet() Returns a zero rate for the outflowingAngularMomentum property for the hotHalo component class.

<double precision> outflowingMass() Returns the default value for the outflowingMass property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> outflowingMassAttributeMatch(<logical>
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the hotHalo class that have the desired attributes for the outflowingMass property

<integer> outflowingMassCount() Compute the count of evolvable quantities in the outflowingMass property of the HotHaloStandard component.

<logical> outflowingMassIsGettable() Returns true if the outflowingMass property is gettable for the hotHalo component class.

<logical> outflowingMassIsSettable() Specify whether the outflowingMass property of the hotHalo component is settable.

<void> outflowingMassRate(<double precision> value) Cumulate to the rate of the outflowingMass property of the HotHaloStandard component.

<double precision> outflowingMassRateGet() Returns a zero rate for the outflowingMass property for the hotHalo component class.

<logical> outflowTrackingIsActive() Return true if the outflowTracking implementation of the hotHalo component is the active choice.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a hotHalo component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a null implementation of the hotHalo component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a null implementation of the hotHalo component.

<void> postOutput(<double precision> time→) Perform post-output processing of a hotHalo component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a null implementation of the hotHalo component.

<void> serializeASCII() Serialize the contents of a null implementation of the hotHalo component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a null implementation of the hotHalo component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a null implementation of the hotHalo component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a null implementation of the hotHalo component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a null implementation of the hotHalo component to XML.

<integer(c_size_t)> sizeOf() Return the size in bytes of a null implementation of the hotHalo component.

<logical> standardIsActive() Return true if the standard implementation of the hotHalo component is the active choice.

<type(abundances)> strippedAbundances() Returns the default value for the strippedAbundances property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> strippedAbundancesAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the strippedAbundances property

<integer> strippedAbundancesCount() Compute the count of evolvable quantities in the strippedAbundances property of the HotHaloStandard component.

<logical> strippedAbundancesIsGettable() Returns true if the strippedAbundances property is gettable for the hotHalo component class.

<logical> strippedAbundancesIsSettable() Specify whether the strippedAbundances property of the hotHalo component is settable.

<void> strippedAbundancesRate(<type(abundances)> value) Cumulate to the rate of the strippedAbundances property of the HotHaloStandard component.

<type(abundances)> strippedAbundancesRateGet() Returns a zero rate for the strippedAbundances property for the hotHalo component class.

<void> strippedAbundancesScale(<type(abundances)> value) Set the scale of the strippedAbundances property of the HotHaloStandard component.

<void> strippedAbundancesSet(<type(abundances)> value) Set the strippedAbundances property of the hotHalo component.

<double precision> strippedMass() Returns the default value for the strippedMass property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> strippedMassAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the strippedMass property

<integer> strippedMassCount() Compute the count of evolvable quantities in the strippedMass property of the HotHaloStandard component.

<logical> strippedMassIsGettable() Returns true if the strippedMass property is gettable for the hotHalo component class.

<logical> strippedMassIsSettable() Specify whether the strippedMass property of the hotHalo component is settable.

<void> strippedMassRate(<double precision> value) Cumulate to the rate of the strippedMass property of the HotHaloStandard component.

<double precision> strippedMassRateGet() Returns a zero rate for the strippedMass property for the hotHalo component class.

<void> strippedMassScale(<double precision> value) Set the scale of the strippedMass property of the HotHaloStandard component.

<void> strippedMassSet(<double precision> value) Set the strippedMass property of the hotHalo component.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<type(abundances)> trackedOutflowAbundances() Returns the default value for the trackedOutflowAbundances property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> trackedOutflowAbundancesAttributeMatch(<l [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the trackedOutflowAbundances property

<integer> trackedOutflowAbundancesCount() Compute the count of evolvable quantities in the trackedOutflowAbundan property of the HotHaloOutflowTracking component.

<logical> trackedOutflowAbundancesIsGettable() Returns true if the trackedOutflowAbundances property is gettable for the hotHalo component class.

<logical> trackedOutflowAbundancesIsSettable() Specify whether the trackedOutflowAbundances property of the hotHalo component is settable.

<void> trackedOutflowAbundancesRate(<type(abundances)> value) Cumulate to the rate of the trackedOutflowAbundances property of the HotHaloOutflowTracking component.

<type(abundances)> trackedOutflowAbundancesRateGet() Returns a zero rate for the trackedOutflowAbundances property for the hotHalo component class.

<void> trackedOutflowAbundancesScale(<type(abundances)> value) Set the scale of the trackedOutflowAbundances property of the HotHaloOutflowTracking component.

<void> trackedOutflowAbundancesSet(<type(abundances)> value) Set the trackedOutflowAbundances property of the hotHalo component.

<double precision> trackedOutflowMass() Returns the default value for the trackedOutflowMass property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> trackedOutflowMassAttributeMatch(<logical [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the trackedOutflowMass property

<integer> trackedOutflowMassCount() Compute the count of evolvable quantities in the trackedOutflowMass property of the HotHaloOutflowTracking component.

<logical> trackedOutflowMassIsGettable() Returns true if the trackedOutflowMass property is gettable for the hotHalo component class.

<logical> trackedOutflowMassIsSettable() Specify whether the trackedOutflowMass property of the hotHalo component is settable.

<void> trackedOutflowMassRate(<double precision> value) Cumulate to the rate of the trackedOutflowMass property of the HotHaloOutflowTracking component.

<double precision> trackedOutflowMassRateGet() Returns a zero rate for the trackedOutflowMass property for the hotHalo component class.

<void> trackedOutflowMassScale(<double precision> value) Set the scale of the trackedOutflowMass property of the HotHaloOutflowTracking component.

<void> trackedOutflowMassSet(<double precision> value) Set the trackedOutflowMass property of the hotHalo component.

<type(varying_string)> type() Returns the type name for the null implementation of the hotHalo component class.

<double precision> unaccretedMass() Returns the default value for the unaccretedMass property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> unaccretedMassAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the unaccretedMass property

<integer> unaccretedMassCount() Compute the count of evolvable quantities in the unaccretedMass property of the HotHaloStandard component.

<logical> unaccretedMassIsGettable() Returns true if the unaccretedMass property is gettable for the hotHalo component class.

<logical> unaccretedMassIsSettable() Specify whether the unaccretedMass property of the hotHalo component is settable.

<void> unaccretedMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interrupt [interruptProcedure]↔) Accept a rate set for the unaccretedMass property of the hotHalo component class. Trigger an interrupt to create the component.

<double precision> unaccretedMassRateGet() Returns a zero rate for the unaccretedMass property for the hotHalo component class.

<void> unaccretedMassScale(<double precision> value) Set the scale of the unaccretedMass property of the HotHaloStandard component.

<void> unaccretedMassSet(<double precision> value) Set the unaccretedMass property of the hotHalo component.

<logical> verySimpleDelayedIsActive() Return true if the verySimpleDelayed implementation of the hotHalo component is the active choice.

<logical> verySimpleIsActive() Return true if the verySimple implementation of the hotHalo component is the active choice.

nodeComponentHotHaloOutflowTracking

<type(abundances)> abundances() Get the abundances property of an standard implementation of the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> abundancesAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the abundances property

<type(abundances)> abundancesCold() Returns the default value for the abundancesCold property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> abundancesColdAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the abundancesCold property

<integer> abundancesColdCount() Compute the count of evolvable quantities in the abundancesCold property of the HotHaloColdMode component.

<logical> abundancesColdIsGettable() Returns true if the abundancesCold property is gettable for the hotHalo component class.

<logical> abundancesColdIsSettable() Specify whether the abundancesCold property of the hotHalo component is settable.

<void> abundancesColdRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interrupt [interruptProcedure]↔) Accept a rate set for the abundancesCold property of the hotHalo component class. Trigger an interrupt to create the component.

<type(abundances)> abundancesColdRateGet() Returns a zero rate for the abundancesCold property for the hotHalo component class.

<void> abundancesColdScale(<type(abundances)> value) Set the scale of the abundancesCold property of the HotHaloColdMode component.

<void> abundancesColdSet(<type(abundances)> value) Set the abundancesCold property of the hotHalo component.

<integer> abundancesCount() Return a count of the number of scalar properties in the abundances property of an standard implementation of the hotHalo component class.

<void> abundancesInactive() Indicate that the abundances property of an standard implementation of the hotHalo component class is inactive for differential equation solving.

<logical> abundancesIsGettable() Returns true if the abundances property is gettable for the hotHalo component class.

<logical> abundancesIsSettable() Specify whether the abundances property of the hotHalo component is settable.

<void> abundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask [interruptProcedure]↔) Accumulate to the rate of change of the abundances property of an standard implementation of the hotHalo component class.

<type(abundances)> abundancesRateGet() Get the rate of change of the abundances property of an standard implementation of the hotHalo component class.

<void> abundancesScale(<type(abundances)> setValue→) Set the absolute scale of the abundances property of an standard implementation of the hotHalo component class.

<void> abundancesSet(<type(abundances)> setValue→) Set the abundances property of an standard implementation of the hotHalo component class.

<double precision> angularMomentum() Get the angularMomentum property of an standard implementation of the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> angularMomentumAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the angularMomentum property

<double precision> angularMomentumCold() Returns the default value for the angularMomentumCold property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> angularMomentumColdAttributeMatch(<logica [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the angularMomentumCold property

<integer> angularMomentumColdCount() Compute the count of evolvable quantities in the angularMomentumCold property of the HotHaloColdMode component.

<logical> angularMomentumColdIsGettable() Returns true if the angularMomentumCold property is gettable for the hotHalo component class.

<logical> angularMomentumColdIsSettable() Specify whether the angularMomentumCold property of the hotHalo component is settable.

<void> angularMomentumColdRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accept a rate set for the angularMomentumCold property of the hotHalo component class. Trigger an interrupt to create the component.

<double precision> angularMomentumColdRateGet() Returns a zero rate for the angularMomentumCold property for the hotHalo component class.

<void> angularMomentumColdScale(<double precision> value) Set the scale of the angularMomentumCold property of the HotHaloColdMode component.

<void> angularMomentumColdSet(<double precision> value) Set the angularMomentumCold property of the hotHalo component.

<integer> angularMomentumCount() Return a count of the number of scalar properties in the angularMomentum property of an standard implementation of the hotHalo component class.

<void> angularMomentumInactive() Indicate that the angularMomentum property of an standard implementation of the hotHalo component class is inactive for differential equation solving.

<logical> angularMomentumIsGettable() Returns true if the angularMomentum property is gettable for the hotHalo component class.

<logical> angularMomentumIsSettable() Specify whether the angularMomentum property of the hotHalo component is settable.

<void> angularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interrup [interruptProcedure]↔) Accumulate to the rate of change of the angularMomentum property of an standard implementation of the hotHalo component class.

<double precision> angularMomentumRateGet() Get the rate of change of the angularMomentum property of an standard implementation of the hotHalo component class.

<void> angularMomentumScale(<double precision> setValue→) Set the absolute scale of the angularMomentum property of an standard implementation of the hotHalo component class.

<void> angularMomentumSet(<double precision> setValue→) Set the angularMomentum property of an standard implementation of the hotHalo component class.

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a outflowTracking implementation of the hotHalo component from a supplied XML definition.

<type(chemicalAbundances)> chemicals() Get the chemicals property of an standard implementation of the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> chemicalsAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the chemicals property

<integer> chemicalsCount() Return a count of the number of scalar properties in the chemicals property of an standard implementation of the hotHalo component class.

<void> chemicalsInactive() Indicate that the chemicals property of an standard implementation of the hotHalo component class is inactive for differential equation solving.

<logical> chemicalsIsGettable() Returns true if the chemicals property is gettable for the hotHalo component class.

<logical> chemicalsIsSettable() Specify whether the chemicals property of the hotHalo component is settable.

<void> chemicalsRate(<type(chemicalAbundances)> setValue→, <logical> [interrupt]↔, <*procedure(interr [interruptProcedure]↔) Accumulate to the rate of change of the chemicals property of an standard implementation of the hotHalo component class.

<type(chemicalAbundances)> chemicalsRateGet() Get the rate of change of the chemicals property of an standard implementation of the hotHalo component class.

<void> chemicalsScale(<type(chemicalAbundances)> setValue→) Set the absolute scale of the chemicals property of an standard implementation of the hotHalo component class.

<void> chemicalsSet(<type(chemicalAbundances)> setValue→) Set the chemicals property of an standard implementation of the hotHalo component class.

<logical> coldModeIsActive() Return true if the coldMode implementation of the hotHalo component is the active choice.

<void> createFunctionSet() Set the create function for the standard implementation of the hotHalo component class.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a outflowTracking implementation of the hotHalo component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a outflowTracking implementation of the hotHalo component from array.

<void> destroy() Finalize a outflowTracking implementation of the hotHalo component.

<void> dumpASCII() Dump the content of a hotHalo component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<double precision> heatSource() Returns the default value for the heatSource property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> heatSourceAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the heatSource property

<integer> heatSourceCount() Compute the count of evolvable quantities in the heatSource property of the HotHaloStandard component.

<logical> heatSourceIsGettable() Returns true if the heatSource property is gettable for the hotHalo component class.

<logical> heatSourceIsSettable() Specify whether the heatSource property of the hotHalo component is settable.

<void> heatSourceRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask [interruptProcedure]↔) Accumulate the rate of change of the heatSource property of the standard implementation of the hotHalo component using a deferred function.

<void> heatSourceRateFunction() Set the function to be used for the rate method of the heatSource property of the HotHaloStandard component.

<double precision> heatSourceRateGet() Returns a zero rate for the heatSource property for the hotHalo component class.

<logical> heatSourceRateIsAttached() Return true if the deferred function used to rate the heatSource property of the HotHaloStandard component class has been attached.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<type(abundances)> hotHaloCoolingAbundances() Returns the default value for the hotHaloCoolingAbundances property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> hotHaloCoolingAbundancesAttributeMatch(<l [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the hotHaloCoolingAbundances property

<integer> hotHaloCoolingAbundancesCount() Compute the count of evolvable quantities in the hotHaloCoolingAbundan property of the HotHaloStandard component.

<logical> hotHaloCoolingAbundancesIsGettable() Returns true if the hotHaloCoolingAbundances property is gettable for the hotHalo component class.

<logical> hotHaloCoolingAbundancesIsSettable() Specify whether the hotHaloCoolingAbundances property of the hotHalo component is settable.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAbundancesRateFunction() Set the function to be used for the rate method of the hotHaloCoolingAbundances property of the hotHalo component.

<type(abundances)> hotHaloCoolingAbundancesRateGet() Returns a zero rate for the hotHaloCoolingAbundances property for the hotHalo component class.

<logical> hotHaloCoolingAbundancesRateIsAttached() Return true if the deferred function used to rate the hotHaloCoolingAbundances property of the hotHalo component class has been attached.

<double precision> hotHaloCoolingAngularMomentum() Returns the default value for the hotHaloCoolingAngularMome property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> hotHaloCoolingAngularMomentumAttributeMat [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the hotHaloCoolingAngularMomentum property

<integer> hotHaloCoolingAngularMomentumCount() Compute the count of evolvable quantities in the hotHaloCoolingAngularMomentum property of the HotHaloStandard component.

<logical> hotHaloCoolingAngularMomentumIsGettable() Returns true if the hotHaloCoolingAngularMomentum property is gettable for the hotHalo component class.

<logical> hotHaloCoolingAngularMomentumIsSettable() Specify whether the hotHaloCoolingAngularMomentum property of the hotHalo component is settable.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRateFunction() Set the function to be used for the rate method of the hotHaloCoolingAngularMomentum property of the hotHalo component.

<double precision> hotHaloCoolingAngularMomentumRateGet() Returns a zero rate for the hotHaloCoolingAngularMo property for the hotHalo component class.

<logical> hotHaloCoolingAngularMomentumRateIsAttached() Return true if the deferred function used to rate the hotHaloCoolingAngularMomentum property of the hotHalo component class has been attached.

<double precision> hotHaloCoolingMass() Returns the default value for the hotHaloCoolingMass property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> hotHaloCoolingMassAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the hotHaloCoolingMass property

<integer> hotHaloCoolingMassCount() Compute the count of evolvable quantities in the hotHaloCoolingMass property of the HotHaloStandard component.

<logical> hotHaloCoolingMassIsGettable() Returns true if the hotHaloCoolingMass property is gettable for the hotHalo component class.

<logical> hotHaloCoolingMassIsSettable() Specify whether the hotHaloCoolingMass property of the hotHalo component is settable.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRateFunction() Set the function to be used for the rate method of the hotHaloCoolingMass property of the hotHalo component.

<double precision> hotHaloCoolingMassRateGet() Returns a zero rate for the hotHaloCoolingMass property for the hotHalo component class.

<logical> hotHaloCoolingMassRateIsAttached() Return true if the deferred function used to rate the hotHaloCoolingMass property of the hotHalo component class has been attached.

<void> initialize() Initialize a outflowTracking member of the hotHalo component.

<logical> isInitialized() Get the isInitialized property of an standard implementation of the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> isInitializedAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the isInitialized property

<logical> isInitializedIsGettable() Returns true if the isInitialized property is gettable for the hotHalo component class.

<logical> isInitializedIsSettable() Specify whether the isInitialized property of the hotHalo component is settable.

<logical> isInitializedRateGet() Returns a zero rate for the isInitialized property for the hotHalo component class.

<void> isInitializedSet(<logical> setValue→) Set the isInitialized property of an standard implementation of the hotHalo component class.

<double precision> mass() Get the mass property of an standard implementation of the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> massAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the mass property

<double precision> massCold() Returns the default value for the massCold property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> massColdAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the massCold property

<integer> massColdCount() Compute the count of evolvable quantities in the massCold property of the HotHaloColdMode component.

<logical> massColdIsGettable() Returns true if the massCold property is gettable for the hotHalo component class.

<logical> massColdIsSettable() Specify whether the massCold property of the hotHalo component is settable.

<void> massColdRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accept a rate set for the massCold property of the hotHalo component class. Trigger an interrupt to create the component.

<double precision> massColdRateGet() Returns a zero rate for the massCold property for the hotHalo component class.

<void> massColdScale(<double precision> value) Set the scale of the massCold property of the HotHaloColdMode component.

<void> massColdSet(<double precision> value) Set the massCold property of the hotHalo component.

<integer> massCount() Return a count of the number of scalar properties in the mass property of an standard implementation of the hotHalo component class.

<void> massInactive() Indicate that the mass property of an standard implementation of the hotHalo component class is inactive for differential equation solving.

<logical> massIsGettable() Returns true if the mass property is gettable for the hotHalo component class.

<logical> massIsSettable() Specify whether the mass property of the hotHalo component is settable.

<void> massRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the mass property of an standard implementation of the hotHalo component class.

<double precision> massRateGet() Get the rate of change of the mass property of an standard implementation of the hotHalo component class.

<void> massRemovalRate() Called whenever the standard hot halo component removes mass from the halo.

<void> massScale(<double precision> setValue→) Set the absolute scale of the mass property of an standard implementation of the hotHalo component class.

<void> massSet(<double precision> setValue→) Set the mass property of an standard implementation of the hotHalo component class.

<double precision> massSink() Returns the default value for the massSink property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> massSinkAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the massSink property

<integer> massSinkCount() Compute the count of evolvable quantities in the massSink property of the HotHaloStandard component.

<logical> massSinkIsGettable() Returns true if the massSink property is gettable for the hotHalo component class.

<logical> massSinkIsSettable() Specify whether the massSink property of the hotHalo component is settable.

<void> massSinkRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the massSink property of the standard implementation of the hotHalo component using a deferred function.

<void> massSinkRateFunction() Set the function to be used for the rate method of the massSink property of the HotHaloStandard component.

<double precision> massSinkRateGet() Returns a zero rate for the massSink property for the hotHalo component class.

<logical> massSinkRateIsAttached() Return true if the deferred function used to rate the massSink property of the HotHaloStandard component class has been attached.

<double precision> massTotal() Returns the default value for the massTotal property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> massTotalAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the massTotal property

<logical> massTotalIsGettable() Returns true if the massTotal property is gettable for the hotHalo component class.

<logical> massTotalIsSettable() Specify whether the massTotal property of the hotHalo component is settable.

<double precision> massTotalRateGet() Returns a zero rate for the massTotal property for the hotHalo component class.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a outflowTracking implementation of the hotHalo component class.

<logical> nullIsActive() Return true if the null implementation of the hotHalo component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<double precision> outerRadius() Get the value of the outerRadius property of the standard implementation of the hotHalo component using a deferred function.

<type(varying_string), allocatable, dimension(:) => matches> outerRadiusAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the outerRadius property

<integer> outerRadiusCount() Return a count of the number of scalar properties in the outerRadius property of an standard implementation of the hotHalo component class.

<void> outerRadiusFunction() Set the function to be used for the get method of the outerRadius property of the HotHaloStandard component.

<double precision> outerRadiusGrowthRate() Call the deferred function for the outerRadiusGrowthRate method of the hotHalo component class if it has been set.

<void> outerRadiusGrowthRateFunction() Set the function to be used for the outerRadiusGrowthRate method of the outflowTracking implementation of the hotHalo component class.

<logical> outerRadiusGrowthRateFunctionIsSet() Return true if the deferred function for the outerRadiusGrowthRate method of the outflowTracking implementation of the hotHalo component class has been set.

<void> outerRadiusInactive() Indicate that the outerRadius property of an standard implementation of the hotHalo component class is inactive for differential equation solving.

<logical> outerRadiusIsAttached() Return true if the deferred function used to get the outerRadius property of the HotHaloStandard component class has been attached.

<logical> outerRadiusIsGettable() Returns true if the outerRadius property is gettable for the hotHalo component class.

<logical> outerRadiusIsSettable() Specify whether the outerRadius property of the hotHalo component is settable.

<void> outerRadiusRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTas [interruptProcedure]↔) Accumulate to the rate of change of the outerRadius property of an standard implementation of the hotHalo component class.

<double precision> outerRadiusRateGet() Get the rate of change of the outerRadius property of an standard implementation of the hotHalo component class.

<void> outerRadiusScale(<double precision> setValue→) Set the absolute scale of the outerRadius property of an standard implementation of the hotHalo component class.

<void> outerRadiusSet(<double precision> setValue→) Set the outerRadius property of an standard implementation of the hotHalo component class.

<double precision> outerRadiusValue() Get the outerRadius property of an standard implementation of the hotHalo component class.

<type(abundances)> outflowedAbundances() Get the outflowedAbundances property of an standard implementation of the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> outflowedAbundancesAttributeMatch(<logica
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the hotHalo class that have the desired attributes for the outflowedAbundances property

<integer> outflowedAbundancesCount() Return a count of the number of scalar properties in the outflowedAbundances property of an standard implementation of the hotHalo component class.

<void> outflowedAbundancesInactive() Indicate that the outflowedAbundances property of an standard implementation of the hotHalo component class is inactive for differential equation solving.

<logical> outflowedAbundancesIsGettable() Returns true if the outflowedAbundances property is gettable for the hotHalo component class.

<logical> outflowedAbundancesIsSettable() Specify whether the outflowedAbundances property of the hotHalo component is settable.

<void> outflowedAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the outflowedAbundances property of an standard implementation of the hotHalo component class.

<type(abundances)> outflowedAbundancesRateGet() Get the rate of change of the outflowedAbundances property of an standard implementation of the hotHalo component class.

<void> outflowedAbundancesScale(<type(abundances)> setValue→) Set the absolute scale of the outflowedAbundances property of an standard implementation of the hotHalo component class.

<void> outflowedAbundancesSet(<type(abundances)> setValue→) Set the outflowedAbundances property of an standard implementation of the hotHalo component class.

<double precision> outflowedAngularMomentum() Get the outflowedAngularMomentum property of an standard implementation of the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> outflowedAngularMomentumAttributeMatch(<l
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the hotHalo class that have the desired attributes for the outflowedAngularMomentum property

<integer> outflowedAngularMomentumCount() Return a count of the number of scalar properties in the outflowedAngularMomentum property of an standard implementation of the hotHalo component class.

<void> outflowedAngularMomentumInactive() Indicate that the outflowedAngularMomentum property of an standard implementation of the hotHalo component class is inactive for differential equation solving.

<logical> outflowedAngularMomentumIsGettable() Returns true if the outflowedAngularMomentum property is gettable for the hotHalo component class.

<logical> outflowedAngularMomentumIsSettable() Specify whether the outflowedAngularMomentum property of the hotHalo component is settable.

<void> outflowedAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the outflowedAngularMomentum property of an standard implementation of the hotHalo component class.

<double precision> outflowedAngularMomentumRateGet() Get the rate of change of the outflowedAngularMomentum property of an standard implementation of the hotHalo component class.

<void> outflowedAngularMomentumScale(<double precision> setValue→) Set the absolute scale of the outflowedAngularMomentum property of an standard implementation of the hotHalo component class.

<void> outflowedAngularMomentumSet(<double precision> setValue→) Set the outflowedAngularMomentum property of an standard implementation of the hotHalo component class.

<double precision> outflowedMass() Get the outflowedMass property of an standard implementation of the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> outflowedMassAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the outflowedMass property

<integer> outflowedMassCount() Return a count of the number of scalar properties in the outflowedMass property of an standard implementation of the hotHalo component class.

<void> outflowedMassInactive() Indicate that the outflowedMass property of an standard implementation of the hotHalo component class is inactive for differential equation solving.

<logical> outflowedMassIsGettable() Returns true if the outflowedMass property is gettable for the hotHalo component class.

<logical> outflowedMassIsSettable() Specify whether the outflowedMass property of the hotHalo component is settable.

<void> outflowedMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptT [interruptProcedure]↔) Accumulate to the rate of change of the outflowedMass property of an standard implementation of the hotHalo component class.

<double precision> outflowedMassRateGet() Get the rate of change of the outflowedMass property of an standard implementation of the hotHalo component class.

<void> outflowedMassScale(<double precision> setValue→) Set the absolute scale of the outflowedMass property of an standard implementation of the hotHalo component class.

<void> outflowedMassSet(<double precision> setValue→) Set the outflowedMass property of an standard implementation of the hotHalo component class.

<type(abundances)> outflowingAbundances() Returns the default value for the outflowingAbundances property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> outflowingAbundancesAttributeMatch(<logic [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the outflowingAbundances property

<integer> outflowingAbundancesCount() Compute the count of evolvable quantities in the outflowingAbundances property of the HotHaloStandard component.

<logical> outflowingAbundancesIsGettable() Returns true if the outflowingAbundances property is gettable for the hotHalo component class.

<logical> outflowingAbundancesIsSettable() Specify whether the outflowingAbundances property of the hotHalo component is settable.

<void> outflowingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the outflowingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> outflowingAbundancesRateFunction() Set the function to be used for the rate method of the outflowingAbundances property of the HotHaloStandard component.

<type(abundances)> outflowingAbundancesRateGet() Returns a zero rate for the outflowingAbundances property for the hotHalo component class.

<logical> outflowingAbundancesRateIsAttached() Return true if the deferred function used to rate the outflowingAbundances property of the HotHaloStandard component class has been attached.

<double precision> outflowingAngularMomentum() Returns the default value for the outflowingAngularMomentum property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> outflowingAngularMomentumAttributeMatch(<
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the hotHalo class that have the desired attributes for the outflowingAngularMomentum property

<integer> outflowingAngularMomentumCount() Compute the count of evolvable quantities in the outflowingAngularMomentum property of the HotHaloStandard component.

<logical> outflowingAngularMomentumIsGettable() Returns true if the outflowingAngularMomentum property is gettable for the hotHalo component class.

<logical> outflowingAngularMomentumIsSettable() Specify whether the outflowingAngularMomentum property of the hotHalo component is settable.

<void> outflowingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the outflowingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> outflowingAngularMomentumRateFunction() Set the function to be used for the rate method of the outflowingAngularMomentum property of the HotHaloStandard component.

<double precision> outflowingAngularMomentumRateGet() Returns a zero rate for the outflowingAngularMomentum property for the hotHalo component class.

<logical> outflowingAngularMomentumRateIsAttached() Return true if the deferred function used to rate the outflowingAngularMomentum property of the HotHaloStandard component class has been attached.

<double precision> outflowingMass() Returns the default value for the outflowingMass property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> outflowingMassAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the outflowingMass property

<integer> outflowingMassCount() Compute the count of evolvable quantities in the outflowingMass property of the HotHaloStandard component.

<logical> outflowingMassIsGettable() Returns true if the outflowingMass property is gettable for the hotHalo component class.

<logical> outflowingMassIsSettable() Specify whether the outflowingMass property of the hotHalo component is settable.

<void> outflowingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interrupt [interruptProcedure]↔) Accumulate the rate of change of the outflowingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> outflowingMassRateFunction() Set the function to be used for the rate method of the outflowingMass property of the HotHaloStandard component.

<double precision> outflowingMassRateGet() Returns a zero rate for the outflowingMass property for the hotHalo component class.

<logical> outflowingMassRateIsAttached() Return true if the deferred function used to rate the outflowingMass property of the HotHaloStandard component class has been attached.

<void> outflowReturn(<logical> interrupt↔, <*procedure(interruptTask)> interruptprocedure↔) Call the deferred function for the outflowReturn method of the hotHalo component class if it has been set.

<void> outflowReturnFunction() Set the function to be used for the outflowReturn method of the outflowTracking implementation of the hotHalo component class.

<logical> outflowReturnFunctionIsSet() Return true if the deferred function for the outflowReturn method of the outflowTracking implementation of the hotHalo component class has been set.

<logical> outflowTrackingIsActive() Return true if the outflowTracking implementation of the hotHalo component is the active choice.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a hotHalo component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a outflowTracking implementation of the hotHalo component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a outflowTracking implementation of the hotHalo component.

<void> postOutput(<double precision> time→) Perform post-output processing for a outflowTracking implementation of the hotHalo component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a outflowTracking implementation of the hotHalo component.

<void> serializeASCII() Serialize the contents of a outflowTracking implementation of the hotHalo component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a outflowTracking implementation of the hotHalo component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a outflowTracking implementation of the hotHalo component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a outflowTracking implementation of the hotHalo component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a outflowTracking implementation of the hotHalo component to XML.

<integer(c_size_t)> sizeOf() Return the size in bytes of a outflowTracking implementation of the hotHalo component.

<logical> standardIsActive() Return true if the standard implementation of the hotHalo component is the active choice.

<type(abundances)> strippedAbundances() Get the strippedAbundances property of an standard implementation of the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> strippedAbundancesAttributeMatch(<logical [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the strippedAbundances property

<integer> strippedAbundancesCount() Return a count of the number of scalar properties in the strippedAbundances property of an standard implementation of the hotHalo component class.

<void> strippedAbundancesInactive() Indicate that the strippedAbundances property of an standard implementation of the hotHalo component class is inactive for differential equation solving.

<logical> strippedAbundancesIsGettable() Returns true if the strippedAbundances property is gettable for the hotHalo component class.

<logical> strippedAbundancesIsSettable() Specify whether the strippedAbundances property of the hotHalo component is settable.

<void> strippedAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate to the rate of change of the strippedAbundances property of an standard implementation of the hotHalo component class.

<type(abundances)> strippedAbundancesRateGet() Get the rate of change of the strippedAbundances property of an standard implementation of the hotHalo component class.

<void> strippedAbundancesScale(<type(abundances)> setValue→) Set the absolute scale of the strippedAbundances property of an standard implementation of the hotHalo component class.

<void> strippedAbundancesSet(<type(abundances)> setValue→) Set the strippedAbundances property of an standard implementation of the hotHalo component class.

<double precision> strippedMass() Get the strippedMass property of an standard implementation of the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> strippedMassAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the strippedMass property

<integer> strippedMassCount() Return a count of the number of scalar properties in the strippedMass property of an standard implementation of the hotHalo component class.

<void> strippedMassInactive() Indicate that the strippedMass property of an standard implementation of the hotHalo component class is inactive for differential equation solving.

<logical> strippedMassIsGettable() Returns true if the strippedMass property is gettable for the hotHalo component class.

<logical> strippedMassIsSettable() Specify whether the strippedMass property of the hotHalo component is settable.

<void> strippedMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTa [interruptProcedure]↔) Accumulate to the rate of change of the strippedMass property of an standard implementation of the hotHalo component class.

<double precision> strippedMassRateGet() Get the rate of change of the strippedMass property of an standard implementation of the hotHalo component class.

<void> strippedMassScale(<double precision> setValue→) Set the absolute scale of the strippedMass property of an standard implementation of the hotHalo component class.

<void> strippedMassSet(<double precision> setValue→) Set the strippedMass property of an standard implementation of the hotHalo component class.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<type(abundances)> trackedOutflowAbundances() Get the trackedOutflowAbundances property of an outflowTracking implementation of the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> trackedOutflowAbundancesAttributeMatch(<l [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the trackedOutflowAbundances property

<integer> trackedOutflowAbundancesCount() Return a count of the number of scalar properties in the trackedOutflowAbundances property of an outflowTracking implementation of the hotHalo component class.

<void> trackedOutflowAbundancesInactive() Indicate that the trackedOutflowAbundances property of an outflowTracking implementation of the hotHalo component class is inactive for differential equation solving.

<logical> trackedOutflowAbundancesIsGettable() Returns true if the trackedOutflowAbundances property is gettable for the hotHalo component class.

<logical> trackedOutflowAbundancesIsSettable() Specify whether the trackedOutflowAbundances property of the hotHalo component is settable.

<void> trackedOutflowAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the trackedOutflowAbundances property of an outflowTracking implementation of the hotHalo component class.

<type(abundances)> trackedOutflowAbundancesRateGet() Get the rate of change of the trackedOutflowAbundances property of an outflowTracking implementation of the hotHalo component class.

<void> trackedOutflowAbundancesScale(<type(abundances)> setValue→) Set the absolute scale of the trackedOutflowAbundances property of an outflowTracking implementation of the hotHalo component class.

<void> trackedOutflowAbundancesSet(<type(abundances)> setValue→) Set the trackedOutflowAbundances property of an outflowTracking implementation of the hotHalo component class.

<double precision> trackedOutflowMass() Get the trackedOutflowMass property of an outflowTracking implementation of the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> trackedOutflowMassAttributeMatch(<logical [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the trackedOutflowMass property

<integer> trackedOutflowMassCount() Return a count of the number of scalar properties in the trackedOutflowMass property of an outflowTracking implementation of the hotHalo component class.

<void> trackedOutflowMassInactive() Indicate that the trackedOutflowMass property of an outflowTracking implementation of the hotHalo component class is inactive for differential equation solving.

<logical> trackedOutflowMassIsGettable() Returns true if the trackedOutflowMass property is gettable for the hotHalo component class.

<logical> trackedOutflowMassIsSettable() Specify whether the trackedOutflowMass property of the hotHalo component is settable.

<void> trackedOutflowMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate to the rate of change of the trackedOutflowMass property of an outflowTracking implementation of the hotHalo component class.

<double precision> trackedOutflowMassRateGet() Get the rate of change of the trackedOutflowMass property of an outflowTracking implementation of the hotHalo component class.

<void> trackedOutflowMassScale(<double precision> setValue→) Set the absolute scale of the trackedOutflowMass property of an outflowTracking implementation of the hotHalo component class.

<void> trackedOutflowMassSet(<double precision> setValue→) Set the trackedOutflowMass property of an outflowTracking implementation of the hotHalo component class.

<type(varying_string)> type() Returns the type name for the outflowTracking implementation of the hotHalo component class.

<double precision> unaccretedMass() Get the unaccretedMass property of an standard implementation of the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> unaccretedMassAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the unaccretedMass property

<integer> unaccretedMassCount() Return a count of the number of scalar properties in the unaccretedMass property of an standard implementation of the hotHalo component class.

<void> unaccretedMassInactive() Indicate that the unaccretedMass property of an standard implementation of the hotHalo component class is inactive for differential equation solving.

<logical> unaccretedMassIsGettable() Returns true if the unaccretedMass property is gettable for the hotHalo component class.

<logical> unaccretedMassIsSettable() Specify whether the unaccretedMass property of the hotHalo component is settable.

<void> unaccretedMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interrupt [interruptProcedure]↔) Accumulate to the rate of change of the unaccretedMass property of an standard implementation of the hotHalo component class.

<double precision> unaccretedMassRateGet() Get the rate of change of the unaccretedMass property of an standard implementation of the hotHalo component class.

<void> unaccretedMassScale(<double precision> setValue→) Set the absolute scale of the unaccretedMass property of an standard implementation of the hotHalo component class.

<void> unaccretedMassSet(<double precision> setValue→) Set the unaccretedMass property of an standard implementation of the hotHalo component class.

<logical> verySimpleDelayedIsActive() Return true if the verySimpleDelayed implementation of the hotHalo component is the active choice.

<logical> verySimpleIsActive() Return true if the verySimple implementation of the hotHalo component is the active choice.

nodeComponentHotHaloStandard

<type(abundances)> abundances() Get the abundances property of an standard implementation of the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> abundancesAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the abundances property

<type(abundances)> abundancesCold() Returns the default value for the abundancesCold property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> abundancesColdAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the abundancesCold property

<integer> abundancesColdCount() Compute the count of evolvable quantities in the abundancesCold property of the HotHaloColdMode component.

<logical> abundancesColdIsGettable() Returns true if the abundancesCold property is gettable for the hotHalo component class.

<logical> abundancesColdIsSettable() Specify whether the abundancesCold property of the hotHalo component is settable.

<void> abundancesColdRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interrupt [interruptProcedure]↔) Accept a rate set for the abundancesCold property of the hotHalo component class. Trigger an interrupt to create the component.

<type(abundances)> abundancesColdRateGet() Returns a zero rate for the abundancesCold property for the hotHalo component class.

<void> abundancesColdScale(<type(abundances)> value) Set the scale of the abundancesCold property of the HotHaloColdMode component.

<void> abundancesColdSet(<type(abundances)> value) Set the abundancesCold property of the hotHalo component.

<integer> abundancesCount() Return a count of the number of scalar properties in the abundances property of an standard implementation of the hotHalo component class.

<void> abundancesInactive() Indicate that the abundances property of an standard implementation of the hotHalo component class is inactive for differential equation solving.

<logical> abundancesIsGettable() Returns true if the abundances property is gettable for the hotHalo component class.

<logical> abundancesIsSettable() Specify whether the abundances property of the hotHalo component is settable.

<void> abundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask [interruptProcedure]↔) Accumulate to the rate of change of the abundances property of an standard implementation of the hotHalo component class.

<type(abundances)> abundancesRateGet() Get the rate of change of the abundances property of an standard implementation of the hotHalo component class.

<void> abundancesScale(<type(abundances)> setValue→) Set the absolute scale of the abundances property of an standard implementation of the hotHalo component class.

<void> abundancesSet(<type(abundances)> setValue→) Set the abundances property of an standard implementation of the hotHalo component class.

<double precision> angularMomentum() Get the angularMomentum property of an standard implementation of the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> angularMomentumAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the angularMomentum property

<double precision> angularMomentumCold() Returns the default value for the angularMomentumCold property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> angularMomentumColdAttributeMatch(<logica [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the angularMomentumCold property

<integer> angularMomentumColdCount() Compute the count of evolvable quantities in the angularMomentumCold property of the HotHaloColdMode component.

<logical> angularMomentumColdIsGettable() Returns true if the angularMomentumCold property is gettable for the hotHalo component class.

<logical> angularMomentumColdIsSettable() Specify whether the angularMomentumCold property of the hotHalo component is settable.

<void> angularMomentumColdRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accept a rate set for the angularMomentumCold property of the hotHalo component class. Trigger an interrupt to create the component.

<double precision> angularMomentumColdRateGet() Returns a zero rate for the angularMomentumCold property for the hotHalo component class.

<void> angularMomentumColdScale(<double precision> value) Set the scale of the angularMomentumCold property of the HotHaloColdMode component.

<void> angularMomentumColdSet(<double precision> value) Set the angularMomentumCold property of the hotHalo component.

<integer> angularMomentumCount() Return a count of the number of scalar properties in the angularMomentum property of an standard implementation of the hotHalo component class.

<void> angularMomentumInactive() Indicate that the angularMomentum property of an standard implementation of the hotHalo component class is inactive for differential equation solving.

<logical> angularMomentumIsGettable() Returns true if the angularMomentum property is gettable for the hotHalo component class.

<logical> angularMomentumIsSettable() Specify whether the angularMomentum property of the hotHalo component is settable.

<void> angularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interrup [interruptProcedure]↔) Accumulate to the rate of change of the angularMomentum property of an standard implementation of the hotHalo component class.

<double precision> angularMomentumRateGet() Get the rate of change of the angularMomentum property of an standard implementation of the hotHalo component class.

<void> angularMomentumScale(<double precision> setValue→) Set the absolute scale of the angularMomentum property of an standard implementation of the hotHalo component class.

<void> angularMomentumSet(<double precision> setValue→) Set the angularMomentum property of an standard implementation of the hotHalo component class.

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a standard implementation of the hotHalo component from a supplied XML definition.

<type(chemicalAbundances)> chemicals() Get the chemicals property of an standard implementation of the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> chemicalsAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the chemicals property

<integer> chemicalsCount() Return a count of the number of scalar properties in the chemicals property of an standard implementation of the hotHalo component class.

<void> chemicalsInactive() Indicate that the chemicals property of an standard implementation of the hotHalo component class is inactive for differential equation solving.

<logical> chemicalsIsGettable() Returns true if the chemicals property is gettable for the hotHalo component class.

<logical> chemicalsIsSettable() Specify whether the chemicals property of the hotHalo component is settable.

<void> chemicalsRate(<type(chemicalAbundances)> setValue→, <logical> [interrupt]↔, <*procedure(interr [interruptProcedure]↔) Accumulate to the rate of change of the chemicals property of an standard implementation of the hotHalo component class.

<type(chemicalAbundances)> chemicalsRateGet() Get the rate of change of the chemicals property of an standard implementation of the hotHalo component class.

<void> chemicalsScale(<type(chemicalAbundances)> setValue→) Set the absolute scale of the chemicals property of an standard implementation of the hotHalo component class.

<void> chemicalsSet(<type(chemicalAbundances)> setValue→) Set the chemicals property of an standard implementation of the hotHalo component class.

<logical> coldModeIsActive() Return true if the coldMode implementation of the hotHalo component is the active choice.

<void> createFunctionSet() Set the create function for the standard implementation of the hotHalo component class.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a standard implementation of the hotHalo component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a standard implementation of the hotHalo component from array.

<void> destroy() Finalize a standard implementation of the hotHalo component.

<void> dumpASCII() Dump the content of a hotHalo component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<double precision> heatSource() Returns the default value for the heatSource property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> heatSourceAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the heatSource property

<integer> heatSourceCount() Compute the count of evolvable quantities in the heatSource property of the HotHaloStandard component.

<logical> heatSourceIsGettable() Returns true if the heatSource property is gettable for the hotHalo component class.

<logical> heatSourceIsSettable() Specify whether the heatSource property of the hotHalo component is settable.

<void> heatSourceRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask [interruptProcedure]↔) Accumulate the rate of change of the heatSource property of the standard implementation of the hotHalo component using a deferred function.

<void> heatSourceRateFunction() Set the function to be used for the rate method of the heatSource property of the HotHaloStandard component.

<double precision> heatSourceRateGet() Returns a zero rate for the heatSource property for the hotHalo component class.

<logical> heatSourceRateIsAttached() Return true if the deferred function used to rate the heatSource property of the HotHaloStandard component class has been attached.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<type(abundances)> hotHaloCoolingAbundances() Returns the default value for the hotHaloCoolingAbundances property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> hotHaloCoolingAbundancesAttributeMatch(<l [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the hotHaloCoolingAbundances property

<integer> hotHaloCoolingAbundancesCount() Compute the count of evolvable quantities in the hotHaloCoolingAbundan property of the HotHaloStandard component.

<logical> hotHaloCoolingAbundancesIsGettable() Returns true if the hotHaloCoolingAbundances property is gettable for the hotHalo component class.

<logical> hotHaloCoolingAbundancesIsSettable() Specify whether the hotHaloCoolingAbundances property of the hotHalo component is settable.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAbundancesRateFunction() Set the function to be used for the rate method of the hotHaloCoolingAbundances property of the hotHalo component.

<type(abundances)> hotHaloCoolingAbundancesRateGet() Returns a zero rate for the hotHaloCoolingAbundances property for the hotHalo component class.

<logical> hotHaloCoolingAbundancesRateIsAttached() Return true if the deferred function used to rate the hotHaloCoolingAbundances property of the hotHalo component class has been attached.

<double precision> hotHaloCoolingAngularMomentum() Returns the default value for the hotHaloCoolingAngularMome property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> hotHaloCoolingAngularMomentumAttributeMat [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the hotHaloCoolingAngularMomentum property

<integer> hotHaloCoolingAngularMomentumCount() Compute the count of evolvable quantities in the hotHaloCoolingAngularMomentum property of the HotHaloStandard component.

<logical> hotHaloCoolingAngularMomentumIsGettable() Returns true if the hotHaloCoolingAngularMomentum property is gettable for the hotHalo component class.

<logical> hotHaloCoolingAngularMomentumIsSettable() Specify whether the hotHaloCoolingAngularMomentum property of the hotHalo component is settable.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRateFunction() Set the function to be used for the rate method of the hotHaloCoolingAngularMomentum property of the hotHalo component.

<double precision> hotHaloCoolingAngularMomentumRateGet() Returns a zero rate for the hotHaloCoolingAngularMo property for the hotHalo component class.

<logical> hotHaloCoolingAngularMomentumRateIsAttached() Return true if the deferred function used to rate the hotHaloCoolingAngularMomentum property of the hotHalo component class has been attached.

<double precision> hotHaloCoolingMass() Returns the default value for the hotHaloCoolingMass property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> hotHaloCoolingMassAttributeMatch(<logical [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the hotHaloCoolingMass property

<integer> hotHaloCoolingMassCount() Compute the count of evolvable quantities in the hotHaloCoolingMass property of the HotHaloStandard component.

<logical> hotHaloCoolingMassIsGettable() Returns true if the hotHaloCoolingMass property is gettable for the hotHalo component class.

<logical> hotHaloCoolingMassIsSettable() Specify whether the hotHaloCoolingMass property of the hotHalo component is settable.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRateFunction() Set the function to be used for the rate method of the hotHaloCoolingMass property of the hotHalo component.

<double precision> hotHaloCoolingMassRateGet() Returns a zero rate for the hotHaloCoolingMass property for the hotHalo component class.

<logical> hotHaloCoolingMassRateIsAttached() Return true if the deferred function used to rate the hotHaloCoolingMass property of the hotHalo component class has been attached.

<void> initialize() Initialize a standard member of the hotHalo component.

<logical> isInitialized() Get the isInitialized property of an standard implementation of the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> isInitializedAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the isInitialized property

<logical> isInitializedIsGettable() Returns true if the isInitialized property is gettable for the hotHalo component class.

<logical> isInitializedIsSettable() Specify whether the isInitialized property of the hotHalo component is settable.

<logical> isInitializedRateGet() Returns a zero rate for the isInitialized property for the hotHalo component class.

<void> isInitializedSet(<logical> setValue→) Set the isInitialized property of an standard implementation of the hotHalo component class.

<double precision> mass() Get the mass property of an standard implementation of the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> massAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the mass property

<double precision> massCold() Returns the default value for the massCold property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> massColdAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the massCold property

<integer> massColdCount() Compute the count of evolvable quantities in the massCold property of the HotHaloColdMode component.

<logical> massColdIsGettable() Returns true if the massCold property is gettable for the hotHalo component class.

<logical> massColdIsSettable() Specify whether the massCold property of the hotHalo component is settable.

<void> massColdRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accept a rate set for the massCold property of the hotHalo component class. Trigger an interrupt to create the component.

<double precision> massColdRateGet() Returns a zero rate for the massCold property for the hotHalo component class.

<void> massColdScale(<double precision> value) Set the scale of the massCold property of the HotHaloColdMode component.

<void> massColdSet(<double precision> value) Set the massCold property of the hotHalo component.

<integer> massCount() Return a count of the number of scalar properties in the mass property of an standard implementation of the hotHalo component class.

<void> massInactive() Indicate that the mass property of an standard implementation of the hotHalo component class is inactive for differential equation solving.

<logical> massIsGettable() Returns true if the mass property is gettable for the hotHalo component class.

<logical> massIsSettable() Specify whether the mass property of the hotHalo component is settable.

<void> massRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the mass property of an standard implementation of the hotHalo component class.

<double precision> massRateGet() Get the rate of change of the mass property of an standard implementation of the hotHalo component class.

<void> massRemovalRate() Called whenever the standard hot halo component removes mass from the halo.

<void> massScale(<double precision> setValue→) Set the absolute scale of the mass property of an standard implementation of the hotHalo component class.

<void> massSet(<double precision> setValue→) Set the mass property of an standard implementation of the hotHalo component class.

<double precision> massSink() Returns the default value for the massSink property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> massSinkAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the massSink property

<integer> massSinkCount() Compute the count of evolvable quantities in the massSink property of the HotHaloStandard component.

<logical> massSinkIsGettable() Returns true if the massSink property is gettable for the hotHalo component class.

<logical> massSinkIsSettable() Specify whether the massSink property of the hotHalo component is settable.

<void> massSinkRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the massSink property of the standard implementation of the hotHalo component using a deferred function.

<void> massSinkRateFunction() Set the function to be used for the rate method of the massSink property of the HotHaloStandard component.

<double precision> massSinkRateGet() Returns a zero rate for the massSink property for the hotHalo component class.

<logical> massSinkRateIsAttached() Return true if the deferred function used to rate the massSink property of the HotHaloStandard component class has been attached.

<double precision> massTotal() Returns the default value for the massTotal property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> massTotalAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the massTotal property

<logical> massTotalIsGettable() Returns true if the massTotal property is gettable for the hotHalo component class.

<logical> massTotalIsSettable() Specify whether the massTotal property of the hotHalo component is settable.

<double precision> massTotalRateGet() Returns a zero rate for the massTotal property for the hotHalo component class.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a standard implementation of the hotHalo component class.

<logical> nullIsActive() Return true if the null implementation of the hotHalo component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<double precision> outerRadius() Get the value of the outerRadius property of the standard implementation of the hotHalo component using a deferred function.

<type(varying_string), allocatable, dimension(:) => matches> outerRadiusAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the outerRadius property

<integer> outerRadiusCount() Return a count of the number of scalar properties in the outerRadius property of an standard implementation of the hotHalo component class.

<void> outerRadiusFunction() Set the function to be used for the get method of the outerRadius property of the HotHaloStandard component.

<double precision> outerRadiusGrowthRate() Call the deferred function for the outerRadiusGrowthRate method of the hotHalo component class if it has been set.

<void> outerRadiusGrowthRateFunction() Set the function to be used for the outerRadiusGrowthRate method of the standard implementation of the hotHalo component class.

<logical> outerRadiusGrowthRateFunctionIsSet() Return true if the deferred function for the outerRadiusGrowthRate method of the standard implementation of the hotHalo component class has been set.

<void> outerRadiusInactive() Indicate that the outerRadius property of an standard implementation of the hotHalo component class is inactive for differential equation solving.

<logical> outerRadiusIsAttached() Return true if the deferred function used to get the outerRadius property of the HotHaloStandard component class has been attached.

<logical> outerRadiusIsGettable() Returns true if the outerRadius property is gettable for the hotHalo component class.

<logical> outerRadiusIsSettable() Specify whether the outerRadius property of the hotHalo component is settable.

<void> outerRadiusRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTas [interruptProcedure]↔) Accumulate to the rate of change of the outerRadius property of an standard implementation of the hotHalo component class.

<double precision> outerRadiusRateGet() Get the rate of change of the outerRadius property of an standard implementation of the hotHalo component class.

<void> outerRadiusScale(<double precision> setValue→) Set the absolute scale of the outerRadius property of an standard implementation of the hotHalo component class.

<void> outerRadiusSet(<double precision> setValue→) Set the outerRadius property of an standard implementation of the hotHalo component class.

<double precision> outerRadiusValue() Get the outerRadius property of an standard implementation of the hotHalo component class.

<type(abundances)> outflowedAbundances() Get the outflowedAbundances property of an standard implementation of the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> outflowedAbundancesAttributeMatch(<logica [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the outflowedAbundances property

<integer> outflowedAbundancesCount() Return a count of the number of scalar properties in the outflowedAbundances property of an standard implementation of the hotHalo component class.

<void> outflowedAbundancesInactive() Indicate that the outflowedAbundances property of an standard implementation of the hotHalo component class is inactive for differential equation solving.

<logical> outflowedAbundancesIsGettable() Returns true if the outflowedAbundances property is gettable for the hotHalo component class.

<logical> outflowedAbundancesIsSettable() Specify whether the outflowedAbundances property of the hotHalo component is settable.

<void> outflowedAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the outflowedAbundances property of an standard implementation of the hotHalo component class.

<type(abundances)> outflowedAbundancesRateGet() Get the rate of change of the outflowedAbundances property of an standard implementation of the hotHalo component class.

<void> outflowedAbundancesScale(<type(abundances)> setValue→) Set the absolute scale of the outflowedAbundances property of an standard implementation of the hotHalo component class.

<void> outflowedAbundancesSet(<type(abundances)> setValue→) Set the outflowedAbundances property of an standard implementation of the hotHalo component class.

<double precision> outflowedAngularMomentum() Get the outflowedAngularMomentum property of an standard implementation of the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> outflowedAngularMomentumAttributeMatch(<l
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the hotHalo class that have the desired attributes for the outflowedAngularMomentum property

<integer> outflowedAngularMomentumCount() Return a count of the number of scalar properties in the outflowedAngularMomentum property of an standard implementation of the hotHalo component class.

<void> outflowedAngularMomentumInactive() Indicate that the outflowedAngularMomentum property of an standard implementation of the hotHalo component class is inactive for differential equation solving.

<logical> outflowedAngularMomentumIsGettable() Returns true if the outflowedAngularMomentum property is gettable for the hotHalo component class.

<logical> outflowedAngularMomentumIsSettable() Specify whether the outflowedAngularMomentum property of the hotHalo component is settable.

<void> outflowedAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the outflowedAngularMomentum property of an standard implementation of the hotHalo component class.

<double precision> outflowedAngularMomentumRateGet() Get the rate of change of the outflowedAngularMomentum property of an standard implementation of the hotHalo component class.

<void> outflowedAngularMomentumScale(<double precision> setValue→) Set the absolute scale of the outflowedAngularMomentum property of an standard implementation of the hotHalo component class.

<void> outflowedAngularMomentumSet(<double precision> setValue→) Set the outflowedAngularMomentum property of an standard implementation of the hotHalo component class.

<double precision> outflowedMass() Get the outflowedMass property of an standard implementation of the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> outflowedMassAttributeMatch(<logical>
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the hotHalo class that have the desired attributes for the outflowedMass property

<integer> outflowedMassCount() Return a count of the number of scalar properties in the outflowedMass property of an standard implementation of the hotHalo component class.

<void> outflowedMassInactive() Indicate that the outflowedMass property of an standard implementation of the hotHalo component class is inactive for differential equation solving.

<logical> outflowedMassIsGettable() Returns true if the outflowedMass property is gettable for the hotHalo component class.

<logical> outflowedMassIsSettable() Specify whether the outflowedMass property of the hotHalo component is settable.

<void> outflowedMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptT
[interruptProcedure]↔) Accumulate to the rate of change of the outflowedMass property of an standard implementation of the hotHalo component class.

<double precision> outflowedMassRateGet() Get the rate of change of the outflowedMass property of an standard implementation of the hotHalo component class.

<void> outflowedMassScale(<double precision> setValue→) Set the absolute scale of the outflowedMass property of an standard implementation of the hotHalo component class.

<void> outflowedMassSet(<double precision> setValue→) Set the outflowedMass property of an standard implementation of the hotHalo component class.

<type(abundances)> outflowingAbundances() Returns the default value for the outflowingAbundances property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> outflowingAbundancesAttributeMatch(<logic
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the hotHalo class that have the desired attributes for the outflowingAbundances property

<integer> outflowingAbundancesCount() Compute the count of evolvable quantities in the outflowingAbundances property of the HotHaloStandard component.

<logical> outflowingAbundancesIsGettable() Returns true if the outflowingAbundances property is gettable for the hotHalo component class.

<logical> outflowingAbundancesIsSettable() Specify whether the outflowingAbundances property of the hotHalo component is settable.

<void> outflowingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔,
<*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the outflowingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> outflowingAbundancesRateFunction() Set the function to be used for the rate method of the outflowingAbundances property of the HotHaloStandard component.

<type(abundances)> outflowingAbundancesRateGet() Returns a zero rate for the outflowingAbundances property for the hotHalo component class.

<logical> outflowingAbundancesRateIsAttached() Return true if the deferred function used to rate the outflowingAbundances property of the HotHaloStandard component class has been attached.

<double precision> outflowingAngularMomentum() Returns the default value for the outflowingAngularMomentum property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> outflowingAngularMomentumAttributeMatch(<
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the hotHalo class that have the desired attributes for the outflowingAngularMomentum property

<integer> outflowingAngularMomentumCount() Compute the count of evolvable quantities in the outflowingAngularMomentum property of the HotHaloStandard component.

<logical> outflowingAngularMomentumIsGettable() Returns true if the outflowingAngularMomentum property is gettable for the hotHalo component class.

<logical> outflowingAngularMomentumIsSettable() Specify whether the outflowingAngularMomentum property of the hotHalo component is settable.

<void> outflowingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the outflowingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> outflowingAngularMomentumRateFunction() Set the function to be used for the rate method of the outflowingAngularMomentum property of the HotHaloStandard component.

<double precision> outflowingAngularMomentumRateGet() Returns a zero rate for the outflowingAngularMomentum property for the hotHalo component class.

<logical> outflowingAngularMomentumRateIsAttached() Return true if the deferred function used to rate the outflowingAngularMomentum property of the HotHaloStandard component class has been attached.

<double precision> outflowingMass() Returns the default value for the outflowingMass property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> outflowingMassAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the outflowingMass property

<integer> outflowingMassCount() Compute the count of evolvable quantities in the outflowingMass property of the HotHaloStandard component.

<logical> outflowingMassIsGettable() Returns true if the outflowingMass property is gettable for the hotHalo component class.

<logical> outflowingMassIsSettable() Specify whether the outflowingMass property of the hotHalo component is settable.

<void> outflowingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interrupt [interruptProcedure]↔) Accumulate the rate of change of the outflowingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> outflowingMassRateFunction() Set the function to be used for the rate method of the outflowingMass property of the HotHaloStandard component.

<double precision> outflowingMassRateGet() Returns a zero rate for the outflowingMass property for the hotHalo component class.

<logical> outflowingMassRateIsAttached() Return true if the deferred function used to rate the outflowingMass property of the HotHaloStandard component class has been attached.

<void> outflowReturn(<logical> interrupt↔, <*procedure(interruptTask)> interruptprocedure↔) Call the deferred function for the outflowReturn method of the hotHalo component class if it has been set.

<void> outflowReturnFunction() Set the function to be used for the outflowReturn method of the standard implementation of the hotHalo component class.

<logical> outflowReturnFunctionIsSet() Return true if the deferred function for the outflowReturn method of the standard implementation of the hotHalo component class has been set.

<logical> outflowTrackingIsActive() Return true if the outflowTracking implementation of the hotHalo component is the active choice.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a hotHalo component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a standard implementation of the hotHalo component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a standard implementation of the hotHalo component.

<void> postOutput(<double precision> time→) Perform post-output processing for a standard implementation of the hotHalo component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a standard implementation of the hotHalo component.

<void> serializeASCII() Serialize the contents of a standard implementation of the hotHalo component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a standard implementation of the hotHalo component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a standard implementation of the hotHalo component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a standard implementation of the hotHalo component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a standard implementation of the hotHalo component to XML.

<integer(c_size_t)> sizeOf() Return the size in bytes of a standard implementation of the hotHalo component.

<logical> standardIsActive() Return true if the standard implementation of the hotHalo component is the active choice.

<type(abundances)> strippedAbundances() Get the strippedAbundances property of an standard implementation of the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> strippedAbundancesAttributeMatch(<logical [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the strippedAbundances property

<integer> strippedAbundancesCount() Return a count of the number of scalar properties in the strippedAbundances property of an standard implementation of the hotHalo component class.

<void> strippedAbundancesInactive() Indicate that the strippedAbundances property of an standard implementation of the hotHalo component class is inactive for differential equation solving.

<logical> strippedAbundancesIsGettable() Returns true if the strippedAbundances property is gettable for the hotHalo component class.

<logical> strippedAbundancesIsSettable() Specify whether the strippedAbundances property of the hotHalo component is settable.

<void> strippedAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate to the rate of change of the strippedAbundances property of an standard implementation of the hotHalo component class.

<type(abundances)> strippedAbundancesRateGet() Get the rate of change of the strippedAbundances property of an standard implementation of the hotHalo component class.

<void> strippedAbundancesScale(<type(abundances)> setValue→) Set the absolute scale of the strippedAbundances property of an standard implementation of the hotHalo component class.

<void> strippedAbundancesSet(<type(abundances)> setValue→) Set the strippedAbundances property of an standard implementation of the hotHalo component class.

<double precision> strippedMass() Get the strippedMass property of an standard implementation of the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> strippedMassAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the strippedMass property

<integer> strippedMassCount() Return a count of the number of scalar properties in the strippedMass property of an standard implementation of the hotHalo component class.

<void> strippedMassInactive() Indicate that the strippedMass property of an standard implementation of the hotHalo component class is inactive for differential equation solving.

<logical> strippedMassIsGettable() Returns true if the strippedMass property is gettable for the hotHalo component class.

<logical> strippedMassIsSettable() Specify whether the strippedMass property of the hotHalo component is settable.

<void> strippedMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTa [interruptProcedure]↔) Accumulate to the rate of change of the strippedMass property of an standard implementation of the hotHalo component class.

<double precision> strippedMassRateGet() Get the rate of change of the strippedMass property of an standard implementation of the hotHalo component class.

<void> strippedMassScale(<double precision> setValue→) Set the absolute scale of the strippedMass property of an standard implementation of the hotHalo component class.

<void> strippedMassSet(<double precision> setValue→) Set the strippedMass property of an standard implementation of the hotHalo component class.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<type(abundances)> trackedOutflowAbundances() Returns the default value for the trackedOutflowAbundances property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> trackedOutflowAbundancesAttributeMatch(<l [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the trackedOutflowAbundances property

<integer> trackedOutflowAbundancesCount() Compute the count of evolvable quantities in the trackedOutflowAbundan property of the HotHaloOutflowTracking component.

<logical> trackedOutflowAbundancesIsGettable() Returns true if the trackedOutflowAbundances property is gettable for the hotHalo component class.

<logical> trackedOutflowAbundancesIsSettable() Specify whether the trackedOutflowAbundances property of the hotHalo component is settable.

<void> trackedOutflowAbundancesRate(<type(abundances)> value) Cumulate to the rate of the trackedOutflowAbundances property of the HotHaloOutflowTracking component.

<type(abundances)> trackedOutflowAbundancesRateGet() Returns a zero rate for the trackedOutflowAbundances property for the hotHalo component class.

<void> trackedOutflowAbundancesScale(<type(abundances)> value) Set the scale of the trackedOutflowAbundances property of the HotHaloOutflowTracking component.

<void> trackedOutflowAbundancesSet(<type(abundances)> value) Set the trackedOutflowAbundances property of the hotHalo component.

<double precision> trackedOutflowMass() Returns the default value for the trackedOutflowMass property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> trackedOutflowMassAttributeMatch(<logical [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the trackedOutflowMass property

<integer> trackedOutflowMassCount() Compute the count of evolvable quantities in the trackedOutflowMass property of the HotHaloOutflowTracking component.

<logical> trackedOutflowMassIsGettable() Returns true if the trackedOutflowMass property is gettable for the hotHalo component class.

<logical> trackedOutflowMassIsSettable() Specify whether the trackedOutflowMass property of the hotHalo component is settable.

<void> trackedOutflowMassRate(<double precision> value) Cumulate to the rate of the trackedOutflowMass property of the HotHaloOutflowTracking component.

<double precision> trackedOutflowMassRateGet() Returns a zero rate for the trackedOutflowMass property for the hotHalo component class.

<void> trackedOutflowMassScale(<double precision> value) Set the scale of the trackedOutflowMass property of the HotHaloOutflowTracking component.

<void> trackedOutflowMassSet(<double precision> value) Set the trackedOutflowMass property of the hotHalo component.

<type(varying_string)> type() Returns the type name for the standard implementation of the hotHalo component class.

<double precision> unaccretedMass() Get the unaccretedMass property of an standard implementation of the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> unaccretedMassAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the unaccretedMass property

<integer> unaccretedMassCount() Return a count of the number of scalar properties in the unaccretedMass property of an standard implementation of the hotHalo component class.

<void> unaccretedMassInactive() Indicate that the unaccretedMass property of an standard implementation of the hotHalo component class is inactive for differential equation solving.

<logical> unaccretedMassIsGettable() Returns true if the unaccretedMass property is gettable for the hotHalo component class.

<logical> unaccretedMassIsSettable() Specify whether the unaccretedMass property of the hotHalo component is settable.

<void> unaccretedMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interrupt [interruptProcedure]↔) Accumulate to the rate of change of the unaccretedMass property of an standard implementation of the hotHalo component class.

<double precision> unaccretedMassRateGet() Get the rate of change of the unaccretedMass property of an standard implementation of the hotHalo component class.

<void> unaccretedMassScale(<double precision> setValue→) Set the absolute scale of the unaccretedMass property of an standard implementation of the hotHalo component class.

<void> unaccretedMassSet(<double precision> setValue→) Set the unaccretedMass property of an standard implementation of the hotHalo component class.

<logical> verySimpleDelayedIsActive() Return true if the verySimpleDelayed implementation of the hotHalo component is the active choice.

<logical> verySimpleIsActive() Return true if the verySimple implementation of the hotHalo component is the active choice.

nodeComponentHotHaloVerySimple

<type(abundances)> abundances() Get the abundances property of an verySimple implementation of the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> abundancesAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the abundances property

<type(abundances)> abundancesCold() Returns the default value for the abundancesCold property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> abundancesColdAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the abundancesCold property

<integer> abundancesColdCount() Compute the count of evolvable quantities in the abundancesCold property of the HotHaloColdMode component.

<logical> abundancesColdIsGettable() Returns true if the abundancesCold property is gettable for the hotHalo component class.

<logical> abundancesColdIsSettable() Specify whether the abundancesCold property of the hotHalo component is settable.

<void> abundancesColdRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interrupt [interruptProcedure]↔) Accept a rate set for the abundancesCold property of the hotHalo component class. Trigger an interrupt to create the component.

<type(abundances)> abundancesColdRateGet() Returns a zero rate for the abundancesCold property for the hotHalo component class.

<void> abundancesColdScale(<type(abundances)> value) Set the scale of the abundancesCold property of the HotHaloColdMode component.

<void> abundancesColdSet(<type(abundances)> value) Set the abundancesCold property of the hotHalo component.

<integer> abundancesCount() Return a count of the number of scalar properties in the abundances property of an verySimple implementation of the hotHalo component class.

<void> abundancesInactive() Indicate that the abundances property of an verySimple implementation of the hotHalo component class is inactive for differential equation solving.

<logical> abundancesIsGettable() Returns true if the abundances property is gettable for the hotHalo component class.

<logical> abundancesIsSettable() Specify whether the abundances property of the hotHalo component is settable.

<void> abundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask [interruptProcedure]↔) Accumulate to the rate of change of the abundances property of an verySimple implementation of the hotHalo component class.

<type(abundances)> abundancesRateGet() Get the rate of change of the abundances property of an verySimple implementation of the hotHalo component class.

<void> abundancesScale(<type(abundances)> setValue→) Set the absolute scale of the abundances property of an verySimple implementation of the hotHalo component class.

<void> abundancesSet(<type(abundances)> setValue→) Set the abundances property of an verySimple implementation of the hotHalo component class.

<double precision> angularMomentum() Returns the default value for the angularMomentum property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> angularMomentumAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the angularMomentum property

<double precision> angularMomentumCold() Returns the default value for the angularMomentumCold property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> angularMomentumColdAttributeMatch(<logica [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the angularMomentumCold property

<integer> angularMomentumColdCount() Compute the count of evolvable quantities in the angularMomentumCold property of the HotHaloColdMode component.

<logical> angularMomentumColdIsGettable() Returns true if the angularMomentumCold property is gettable for the hotHalo component class.

<logical> angularMomentumColdIsSettable() Specify whether the angularMomentumCold property of the hotHalo component is settable.

<void> angularMomentumColdRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accept a rate set for the angularMomentumCold property of the hotHalo component class. Trigger an interrupt to create the component.

<double precision> angularMomentumColdRateGet() Returns a zero rate for the angularMomentumCold property for the hotHalo component class.

<void> angularMomentumColdScale(<double precision> value) Set the scale of the angularMomentumCold property of the HotHaloColdMode component.

<void> angularMomentumColdSet(<double precision> value) Set the angularMomentumCold property of the hotHalo component.

<integer> angularMomentumCount() Compute the count of evolvable quantities in the angularMomentum property of the HotHaloStandard component.

<logical> angularMomentumIsGettable() Returns true if the angularMomentum property is gettable for the hotHalo component class.

<logical> angularMomentumIsSettable() Specify whether the angularMomentum property of the hotHalo component is settable.

<void> angularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interrup [interruptProcedure]↔) Accept a rate set for the angularMomentum property of the hotHalo component class. Trigger an interrupt to create the component.

<double precision> angularMomentumRateGet() Returns a zero rate for the angularMomentum property for the hotHalo component class.

<void> angularMomentumScale(<double precision> value) Set the scale of the angularMomentum property of the HotHaloStandard component.

<void> angularMomentumSet(<double precision> value) Set the angularMomentum property of the hotHalo component.

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a verySimple implementation of the hotHalo component from a supplied XML definition.

<type(chemicalAbundances)> chemicals() Returns the default value for the chemicals property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> chemicalsAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the chemicals property

<integer> chemicalsCount() Compute the count of evolvable quantities in the chemicals property of the HotHaloStandard component.

<logical> chemicalsIsGettable() Returns true if the chemicals property is gettable for the hotHalo component class.

<logical> chemicalsIsSettable() Specify whether the chemicals property of the hotHalo component is settable.

<void> chemicalsRate(<type(chemicalAbundances)> setValue→, <logical> [interrupt]↔, <*procedure(interr [interruptProcedure]↔) Accept a rate set for the chemicals property of the hotHalo component class. Trigger an interrupt to create the component.

<type(chemicalAbundances)> chemicalsRateGet() Returns a zero rate for the chemicals property for the hotHalo component class.

<void> chemicalsScale(<type(chemicalAbundances)> value) Set the scale of the chemicals property of the HotHaloStandard component.

<void> chemicalsSet(<type(chemicalAbundances)> value) Set the chemicals property of the hotHalo component.

<logical> coldModeIsActive() Return true if the coldMode implementation of the hotHalo component is the active choice.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a verySimple implementation of the hotHalo component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a verySimple implementation of the hotHalo component from array.

<void> destroy() Finalize a verySimple implementation of the hotHalo component.

<void> dumpASCII() Dump the content of a hotHalo component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<double precision> heatSource() Returns the default value for the heatSource property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> heatSourceAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the heatSource property

<integer> heatSourceCount() Compute the count of evolvable quantities in the heatSource property of the HotHaloStandard component.

<logical> heatSourceIsGettable() Returns true if the heatSource property is gettable for the hotHalo component class.

<logical> heatSourceIsSettable() Specify whether the heatSource property of the hotHalo component is settable.

<void> heatSourceRate(<double precision> value) Cumulate to the rate of the heatSource property of the HotHaloStandard component.

<double precision> heatSourceRateGet() Returns a zero rate for the heatSource property for the hotHalo component class.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<type(abundances)> hotHaloCoolingAbundances() Returns the default value for the hotHaloCoolingAbundances property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> hotHaloCoolingAbundancesAttributeMatch(<l [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the hotHaloCoolingAbundances property

<integer> hotHaloCoolingAbundancesCount() Compute the count of evolvable quantities in the hotHaloCoolingAbundan property of the HotHaloStandard component.

<logical> hotHaloCoolingAbundancesIsGettable() Returns true if the hotHaloCoolingAbundances property is gettable for the hotHalo component class.

<logical> hotHaloCoolingAbundancesIsSettable() Specify whether the hotHaloCoolingAbundances property of the hotHalo component is settable.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAbundancesRateFunction() Set the function to be used for the rate method of the hotHaloCoolingAbundances property of the hotHalo component.

<type(abundances)> hotHaloCoolingAbundancesRateGet() Returns a zero rate for the hotHaloCoolingAbundances property for the hotHalo component class.

<logical> hotHaloCoolingAbundancesRateIsAttached() Return true if the deferred function used to rate the hotHaloCoolingAbundances property of the hotHalo component class has been attached.

<double precision> hotHaloCoolingAngularMomentum() Returns the default value for the hotHaloCoolingAngularMome property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> hotHaloCoolingAngularMomentumAttributeMat [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the hotHaloCoolingAngularMomentum property

<integer> hotHaloCoolingAngularMomentumCount() Compute the count of evolvable quantities in the hotHaloCoolingAngularMomentum property of the HotHaloStandard component.

<logical> hotHaloCoolingAngularMomentumIsGettable() Returns true if the hotHaloCoolingAngularMomentum property is gettable for the hotHalo component class.

<logical> hotHaloCoolingAngularMomentumIsSettable() Specify whether the hotHaloCoolingAngularMomentum property of the hotHalo component is settable.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRateFunction() Set the function to be used for the rate method of the hotHaloCoolingAngularMomentum property of the hotHalo component.

<double precision> hotHaloCoolingAngularMomentumRateGet() Returns a zero rate for the hotHaloCoolingAngularMo property for the hotHalo component class.

<logical> hotHaloCoolingAngularMomentumRateIsAttached() Return true if the deferred function used to rate the hotHaloCoolingAngularMomentum property of the hotHalo component class has been attached.

<double precision> hotHaloCoolingMass() Returns the default value for the hotHaloCoolingMass property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> hotHaloCoolingMassAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the hotHaloCoolingMass property

<integer> hotHaloCoolingMassCount() Compute the count of evolvable quantities in the hotHaloCoolingMass property of the HotHaloStandard component.

<logical> hotHaloCoolingMassIsGettable() Returns true if the hotHaloCoolingMass property is gettable for the hotHalo component class.

<logical> hotHaloCoolingMassIsSettable() Specify whether the hotHaloCoolingMass property of the hotHalo component is settable.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRateFunction() Set the function to be used for the rate method of the hotHaloCoolingMass property of the hotHalo component.

<double precision> hotHaloCoolingMassRateGet() Returns a zero rate for the hotHaloCoolingMass property for the hotHalo component class.

<logical> hotHaloCoolingMassRateIsAttached() Return true if the deferred function used to rate the hotHaloCoolingMass property of the hotHalo component class has been attached.

<void> initialize() Initialize a verySimple member of the hotHalo component.

<logical> isInitialized() Returns the default value for the isInitialized property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> isInitializedAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the isInitialized property

<logical> isInitializedIsGettable() Returns true if the isInitialized property is gettable for the hotHalo component class.

<logical> isInitializedIsSettable() Specify whether the isInitialized property of the hotHalo component is settable.

<logical> isInitializedRateGet() Returns a zero rate for the isInitialized property for the hotHalo component class.

<void> isInitializedSet(<logical> value) Set the isInitialized property of the hotHalo component.

<double precision> mass() Get the mass property of an verySimple implementation of the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> massAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the mass property

<double precision> massCold() Returns the default value for the massCold property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> massColdAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the massCold property

<integer> massColdCount() Compute the count of evolvable quantities in the massCold property of the HotHaloColdMode component.

<logical> massColdIsGettable() Returns true if the massCold property is gettable for the hotHalo component class.

<logical> massColdIsSettable() Specify whether the massCold property of the hotHalo component is settable.

<void> massColdRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accept a rate set for the massCold property of the hotHalo component class. Trigger an interrupt to create the component.

<double precision> massColdRateGet() Returns a zero rate for the massCold property for the hotHalo component class.

<void> massColdScale(<double precision> value) Set the scale of the massCold property of the HotHaloColdMode component.

<void> massColdSet(<double precision> value) Set the massCold property of the hotHalo component.

<integer> massCount() Return a count of the number of scalar properties in the mass property of an verySimple implementation of the hotHalo component class.

<void> massInactive() Indicate that the mass property of an verySimple implementation of the hotHalo component class is inactive for differential equation solving.

<logical> massIsGettable() Returns true if the mass property is gettable for the hotHalo component class.

<logical> massIsSettable() Specify whether the mass property of the hotHalo component is settable.

<void> massRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the mass property of an verySimple implementation of the hotHalo component class.

<double precision> massRateGet() Get the rate of change of the mass property of an verySimple implementation of the hotHalo component class.

<void> massScale(<double precision> setValue→) Set the absolute scale of the mass property of an verySimple implementation of the hotHalo component class.

<void> massSet(<double precision> setValue→) Set the mass property of an verySimple implementation of the hotHalo component class.

<double precision> massSink() Returns the default value for the massSink property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> massSinkAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the massSink property

<integer> massSinkCount() Compute the count of evolvable quantities in the massSink property of the HotHaloStandard component.

<logical> massSinkIsGettable() Returns true if the massSink property is gettable for the hotHalo component class.

<logical> massSinkIsSettable() Specify whether the massSink property of the hotHalo component is settable.

<void> massSinkRate(<double precision> value) Cumulate to the rate of the massSink property of the HotHaloStandard component.

<double precision> massSinkRateGet() Returns a zero rate for the massSink property for the hotHalo component class.

<double precision> massTotal() Returns the default value for the massTotal property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> massTotalAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the massTotal property

<logical> massTotalIsGettable() Returns true if the massTotal property is gettable for the hotHalo component class.

<logical> massTotalIsSettable() Specify whether the massTotal property of the hotHalo component is settable.

<double precision> massTotalRateGet() Returns a zero rate for the massTotal property for the hotHalo component class.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a verySimple implementation of the hotHalo component class.

<logical> nullIsActive() Return true if the null implementation of the hotHalo component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<double precision> outerRadius() Get the value of the outerRadius property of the verySimple implementation of the hotHalo component using a deferred function.

<type(varying_string), allocatable, dimension(:) => matches> outerRadiusAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the outerRadius property

<integer> outerRadiusCount() Compute the count of evolvable quantities in the outerRadius property of the HotHaloStandard component.

<void> outerRadiusFunction() Set the function to be used for the get method of the outerRadius property of the HotHaloVerySimple component.

<logical> outerRadiusIsAttached() Return true if the deferred function used to get the outerRadius property of the HotHaloVerySimple component class has been attached.

<logical> outerRadiusIsGettable() Returns true if the outerRadius property is gettable for the hotHalo component class.

<logical> outerRadiusIsSettable() Specify whether the outerRadius property of the hotHalo component is settable.

<void> outerRadiusRate(<double precision> value) Cumulate to the rate of the outerRadius property of the HotHaloStandard component.

<double precision> outerRadiusRateGet() Returns a zero rate for the outerRadius property for the hotHalo component class.

<void> outerRadiusScale(<double precision> value) Set the scale of the outerRadius property of the HotHaloStandard component.

<void> outerRadiusSet(<double precision> value) Set the outerRadius property of the hotHalo component.

<type(abundances)> outflowedAbundances() Returns the default value for the outflowedAbundances property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> outflowedAbundancesAttributeMatch(<logica
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the hotHalo class that have the desired attributes for the outflowedAbundances property

<integer> outflowedAbundancesCount() Compute the count of evolvable quantities in the outflowedAbundances property of the HotHaloStandard component.

<logical> outflowedAbundancesIsGettable() Returns true if the outflowedAbundances property is gettable for the hotHalo component class.

<logical> outflowedAbundancesIsSettable() Specify whether the outflowedAbundances property of the hotHalo component is settable.

<void> outflowedAbundancesRate(<type(abundances)> value) Cumulate to the rate of the outflowedAbundances property of the HotHaloStandard component.

<type(abundances)> outflowedAbundancesRateGet() Returns a zero rate for the outflowedAbundances property for the hotHalo component class.

<void> outflowedAbundancesScale(<type(abundances)> value) Set the scale of the outflowedAbundances property of the HotHaloStandard component.

<void> outflowedAbundancesSet(<type(abundances)> value) Set the outflowedAbundances property of the hotHalo component.

<double precision> outflowedAngularMomentum() Returns the default value for the outflowedAngularMomentum property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> outflowedAngularMomentumAttributeMatch(<l [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the outflowedAngularMomentum property

<integer> outflowedAngularMomentumCount() Compute the count of evolvable quantities in the outflowedAngularMomen property of the HotHaloStandard component.

<logical> outflowedAngularMomentumIsGettable() Returns true if the outflowedAngularMomentum property is gettable for the hotHalo component class.

<logical> outflowedAngularMomentumIsSettable() Specify whether the outflowedAngularMomentum property of the hotHalo component is settable.

<void> outflowedAngularMomentumRate(<double precision> value) Cumulate to the rate of the outflowedAngularMomentum property of the HotHaloStandard component.

<double precision> outflowedAngularMomentumRateGet() Returns a zero rate for the outflowedAngularMomentum property for the hotHalo component class.

<void> outflowedAngularMomentumScale(<double precision> value) Set the scale of the outflowedAngularMomentum property of the HotHaloStandard component.

<void> outflowedAngularMomentumSet(<double precision> value) Set the outflowedAngularMomentum property of the hotHalo component.

<double precision> outflowedMass() Returns the default value for the outflowedMass property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> outflowedMassAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the outflowedMass property

<integer> outflowedMassCount() Compute the count of evolvable quantities in the outflowedMass property of the HotHaloStandard component.

<logical> outflowedMassIsGettable() Returns true if the outflowedMass property is gettable for the hotHalo component class.

<logical> outflowedMassIsSettable() Specify whether the outflowedMass property of the hotHalo component is settable.

<void> outflowedMassRate(<double precision> value) Cumulate to the rate of the outflowedMass property of the HotHaloStandard component.

<double precision> outflowedMassRateGet() Returns a zero rate for the outflowedMass property for the hotHalo component class.

<void> outflowedMassScale(<double precision> value) Set the scale of the outflowedMass property of the HotHaloStandard component.

<void> outflowedMassSet(<double precision> value) Set the outflowedMass property of the hotHalo component.

<type(abundances)> outflowingAbundances() Returns the default value for the outflowingAbundances property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> outflowingAbundancesAttributeMatch(<logic [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the outflowingAbundances property

<integer> outflowingAbundancesCount() Compute the count of evolvable quantities in the outflowingAbundances property of the HotHaloStandard component.

<logical> outflowingAbundancesIsGettable() Returns true if the outflowingAbundances property is gettable for the hotHalo component class.

<logical> outflowingAbundancesIsSettable() Specify whether the outflowingAbundances property of the hotHalo component is settable.

<void> outflowingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the outflowingAbundances property of the verySimple implementation of the hotHalo component using a deferred function.

<void> outflowingAbundancesRateFunction() Set the function to be used for the rate method of the outflowingAbundances property of the HotHaloVerySimple component.

<type(abundances)> outflowingAbundancesRateGet() Returns a zero rate for the outflowingAbundances property for the hotHalo component class.

<logical> outflowingAbundancesRateIsAttached() Return true if the deferred function used to rate the outflowingAbundances property of the HotHaloVerySimple component class has been attached.

<double precision> outflowingAngularMomentum() Returns the default value for the outflowingAngularMomentum property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> outflowingAngularMomentumAttributeMatch(< [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the outflowingAngularMomentum property

<integer> outflowingAngularMomentumCount() Compute the count of evolvable quantities in the outflowingAngularMomentum property of the HotHaloStandard component.

<logical> outflowingAngularMomentumIsGettable() Returns true if the outflowingAngularMomentum property is gettable for the hotHalo component class.

<logical> outflowingAngularMomentumIsSettable() Specify whether the outflowingAngularMomentum property of the hotHalo component is settable.

<void> outflowingAngularMomentumRate(<double precision> value) Cumulate to the rate of the outflowingAngularMomentum property of the HotHaloStandard component.

<double precision> outflowingAngularMomentumRateGet() Returns a zero rate for the outflowingAngularMomentum property for the hotHalo component class.

<double precision> outflowingMass() Returns the default value for the outflowingMass property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> outflowingMassAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the outflowingMass property

<integer> outflowingMassCount() Compute the count of evolvable quantities in the outflowingMass property of the HotHaloStandard component.

<logical> outflowingMassIsGettable() Returns true if the outflowingMass property is gettable for the hotHalo component class.

<logical> outflowingMassIsSettable() Specify whether the outflowingMass property of the hotHalo component is settable.

<void> outflowingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interrupt [interruptProcedure]↔) Accumulate the rate of change of the outflowingMass property of the verySimple implementation of the hotHalo component using a deferred function.

<void> outflowingMassRateFunction() Set the function to be used for the rate method of the outflowingMass property of the HotHaloVerySimple component.

<double precision> outflowingMassRateGet() Returns a zero rate for the outflowingMass property for the hotHalo component class.

<logical> outflowingMassRateIsAttached() Return true if the deferred function used to rate the outflowingMass property of the HotHaloVerySimple component class has been attached.

<logical> outflowTrackingIsActive() Return true if the outflowTracking implementation of the hotHalo component is the active choice.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a hotHalo component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a verySimple implementation of the hotHalo component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a verySimple implementation of the hotHalo component.

<void> postOutput(<double precision> time→) Perform post-output processing for a verySimple implementation of the hotHalo component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a verySimple implementation of the hotHalo component.

<void> serializeASCII() Serialize the contents of a verySimple implementation of the hotHalo component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a verySimple implementation of the hotHalo component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a verySimple implementation of the hotHalo component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a verySimple implementation of the hotHalo component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a verySimple implementation of the hotHalo component to XML.

<integer(c_size_t)> sizeOf() Return the size in bytes of a verySimple implementation of the hotHalo component.

<logical> standardIsActive() Return true if the standard implementation of the hotHalo component is the active choice.

<type(abundances)> strippedAbundances() Returns the default value for the strippedAbundances property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> strippedAbundancesAttributeMatch(<logical [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the strippedAbundances property

<integer> strippedAbundancesCount() Compute the count of evolvable quantities in the strippedAbundances property of the HotHaloStandard component.

<logical> strippedAbundancesIsGettable() Returns true if the strippedAbundances property is gettable for the hotHalo component class.

<logical> strippedAbundancesIsSettable() Specify whether the strippedAbundances property of the hotHalo component is settable.

<void> strippedAbundancesRate(<type(abundances)> value) Cumulate to the rate of the strippedAbundances property of the HotHaloStandard component.

<type(abundances)> strippedAbundancesRateGet() Returns a zero rate for the strippedAbundances property for the hotHalo component class.

<void> strippedAbundancesScale(<type(abundances)> value) Set the scale of the strippedAbundances property of the HotHaloStandard component.

<void> strippedAbundancesSet(<type(abundances)> value) Set the strippedAbundances property of the hotHalo component.

<double precision> strippedMass() Returns the default value for the strippedMass property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> strippedMassAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the strippedMass property

<integer> strippedMassCount() Compute the count of evolvable quantities in the strippedMass property of the HotHaloStandard component.

<logical> strippedMassIsGettable() Returns true if the strippedMass property is gettable for the hotHalo component class.

<logical> strippedMassIsSettable() Specify whether the strippedMass property of the hotHalo component is settable.

<void> strippedMassRate(<double precision> value) Cumulate to the rate of the strippedMass property of the HotHaloStandard component.

<double precision> strippedMassRateGet() Returns a zero rate for the strippedMass property for the hotHalo component class.

<void> strippedMassScale(<double precision> value) Set the scale of the strippedMass property of the HotHaloStandard component.

<void> strippedMassSet(<double precision> value) Set the strippedMass property of the hotHalo component.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<type(abundances)> trackedOutflowAbundances() Returns the default value for the trackedOutflowAbundances property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> trackedOutflowAbundancesAttributeMatch(<l [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the trackedOutflowAbundances property

<integer> trackedOutflowAbundancesCount() Compute the count of evolvable quantities in the trackedOutflowAbundan property of the HotHaloOutflowTracking component.

<logical> trackedOutflowAbundancesIsGettable() Returns true if the trackedOutflowAbundances property is gettable for the hotHalo component class.

<logical> trackedOutflowAbundancesIsSettable() Specify whether the trackedOutflowAbundances property of the hotHalo component is settable.

<void> trackedOutflowAbundancesRate(<type(abundances)> value) Cumulate to the rate of the trackedOutflowAbundances property of the HotHaloOutflowTracking component.

<type(abundances)> trackedOutflowAbundancesRateGet() Returns a zero rate for the trackedOutflowAbundances property for the hotHalo component class.

<void> trackedOutflowAbundancesScale(<type(abundances)> value) Set the scale of the trackedOutflowAbundances property of the HotHaloOutflowTracking component.

<void> trackedOutflowAbundancesSet(<type(abundances)> value) Set the trackedOutflowAbundances property of the hotHalo component.

<double precision> trackedOutflowMass() Returns the default value for the trackedOutflowMass property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> trackedOutflowMassAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the trackedOutflowMass property

<integer> trackedOutflowMassCount() Compute the count of evolvable quantities in the trackedOutflowMass property of the HotHaloOutflowTracking component.

<logical> trackedOutflowMassIsGettable() Returns true if the trackedOutflowMass property is gettable for the hotHalo component class.

<logical> trackedOutflowMassIsSettable() Specify whether the trackedOutflowMass property of the hotHalo component is settable.

<void> trackedOutflowMassRate(<double precision> value) Cumulate to the rate of the trackedOutflowMass property of the HotHaloOutflowTracking component.

<double precision> trackedOutflowMassRateGet() Returns a zero rate for the trackedOutflowMass property for the hotHalo component class.

<void> trackedOutflowMassScale(<double precision> value) Set the scale of the trackedOutflowMass property of the HotHaloOutflowTracking component.

<void> trackedOutflowMassSet(<double precision> value) Set the trackedOutflowMass property of the hotHalo component.

<type(varying_string)> type() Returns the type name for the verySimple implementation of the hotHalo component class.

<double precision> unaccretedMass() Get the unaccretedMass property of an verySimple implementation of the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> unaccretedMassAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the unaccretedMass property

<integer> unaccretedMassCount() Return a count of the number of scalar properties in the unaccretedMass property of an verySimple implementation of the hotHalo component class.

<void> unaccretedMassInactive() Indicate that the unaccretedMass property of an verySimple implementation of the hotHalo component class is inactive for differential equation solving.

<logical> unaccretedMassIsGettable() Returns true if the unaccretedMass property is gettable for the hotHalo component class.

<logical> unaccretedMassIsSettable() Specify whether the unaccretedMass property of the hotHalo component is settable.

<void> unaccretedMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interrupt [interruptProcedure]↔) Accumulate to the rate of change of the unaccretedMass property of an verySimple implementation of the hotHalo component class.

<double precision> unaccretedMassRateGet() Get the rate of change of the unaccretedMass property of an verySimple implementation of the hotHalo component class.

<void> unaccretedMassScale(<double precision> setValue→) Set the absolute scale of the unaccretedMass property of an verySimple implementation of the hotHalo component class.

<void> unaccretedMassSet(<double precision> setValue→) Set the unaccretedMass property of an verySimple implementation of the hotHalo component class.

<logical> verySimpleDelayedIsActive() Return true if the verySimpleDelayed implementation of the hotHalo component is the active choice.

<logical> verySimpleIsActive() Return true if the verySimple implementation of the hotHalo component is the active choice.

nodeComponentHotHaloVerySimpleDelayed

<type(abundances)> abundances() Get the abundances property of an verySimple implementation of the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> abundancesAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the abundances property

<type(abundances)> abundancesCold() Returns the default value for the abundancesCold property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> abundancesColdAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the abundancesCold property

<integer> abundancesColdCount() Compute the count of evolvable quantities in the abundancesCold property of the HotHaloColdMode component.

<logical> abundancesColdIsGettable() Returns true if the abundancesCold property is gettable for the hotHalo component class.

<logical> abundancesColdIsSettable() Specify whether the abundancesCold property of the hotHalo component is settable.

<void> abundancesColdRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interrupt
[interruptProcedure]↔) Accept a rate set for the abundancesCold property of the hotHalo component class. Trigger an interrupt to create the component.

<type(abundances)> abundancesColdRateGet() Returns a zero rate for the abundancesCold property for the hotHalo component class.

<void> abundancesColdScale(<type(abundances)> value) Set the scale of the abundancesCold property of the HotHaloColdMode component.

<void> abundancesColdSet(<type(abundances)> value) Set the abundancesCold property of the hotHalo component.

<integer> abundancesCount() Return a count of the number of scalar properties in the abundances property of an verySimple implementation of the hotHalo component class.

<void> abundancesInactive() Indicate that the abundances property of an verySimple implementation of the hotHalo component class is inactive for differential equation solving.

<logical> abundancesIsGettable() Returns true if the abundances property is gettable for the hotHalo component class.

<logical> abundancesIsSettable() Specify whether the abundances property of the hotHalo component is settable.

<void> abundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask
[interruptProcedure]↔) Accumulate to the rate of change of the abundances property of an verySimple implementation of the hotHalo component class.

<type(abundances)> abundancesRateGet() Get the rate of change of the abundances property of an verySimple implementation of the hotHalo component class.

<void> abundancesScale(<type(abundances)> setValue→) Set the absolute scale of the abundances property of an verySimple implementation of the hotHalo component class.

<void> abundancesSet(<type(abundances)> setValue→) Set the abundances property of an verySimple implementation of the hotHalo component class.

<double precision> angularMomentum() Returns the default value for the angularMomentum property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> angularMomentumAttributeMatch(<logical>
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the hotHalo class that have the desired attributes for the angularMomentum property

<double precision> angularMomentumCold() Returns the default value for the angularMomentumCold property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> angularMomentumColdAttributeMatch(<logica
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the hotHalo class that have the desired attributes for the angularMomentumCold property

<integer> angularMomentumColdCount() Compute the count of evolvable quantities in the angularMomentumCold property of the HotHaloColdMode component.

<logical> angularMomentumColdIsGettable() Returns true if the angularMomentumCold property is gettable for the hotHalo component class.

<logical> angularMomentumColdIsSettable() Specify whether the angularMomentumCold property of the hotHalo component is settable.

<void> angularMomentumColdRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accept a rate set for the angularMomentumCold property of the hotHalo component class. Trigger an interrupt to create the component.

<double precision> angularMomentumColdRateGet() Returns a zero rate for the angularMomentumCold property for the hotHalo component class.

<void> angularMomentumColdScale(<double precision> value) Set the scale of the angularMomentumCold property of the HotHaloColdMode component.

<void> angularMomentumColdSet(<double precision> value) Set the angularMomentumCold property of the hotHalo component.

<integer> angularMomentumCount() Compute the count of evolvable quantities in the angularMomentum property of the HotHaloStandard component.

<logical> angularMomentumIsGettable() Returns true if the angularMomentum property is gettable for the hotHalo component class.

<logical> angularMomentumIsSettable() Specify whether the angularMomentum property of the hotHalo component is settable.

<void> angularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interrup [interruptProcedure]↔) Accept a rate set for the angularMomentum property of the hotHalo component class. Trigger an interrupt to create the component.

<double precision> angularMomentumRateGet() Returns a zero rate for the angularMomentum property for the hotHalo component class.

<void> angularMomentumScale(<double precision> value) Set the scale of the angularMomentum property of the HotHaloStandard component.

<void> angularMomentumSet(<double precision> value) Set the angularMomentum property of the hotHalo component.

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a verySimpleDelayed implementation of the hotHalo component from a supplied XML definition.

<type(chemicalAbundances)> chemicals() Returns the default value for the chemicals property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> chemicalsAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the chemicals property

<integer> chemicalsCount() Compute the count of evolvable quantities in the chemicals property of the HotHaloStandard component.

<logical> chemicalsIsGettable() Returns true if the chemicals property is gettable for the hotHalo component class.

<logical> chemicalsIsSettable() Specify whether the chemicals property of the hotHalo component is settable.

<void> chemicalsRate(<type(chemicalAbundances)> setValue→, <logical> [interrupt]↔, <*procedure(interr
[interruptProcedure]↔) Accept a rate set for the chemicals property of the hotHalo component class. Trigger an interrupt to create the component.

<type(chemicalAbundances)> chemicalsRateGet() Returns a zero rate for the chemicals property for the hotHalo component class.

<void> chemicalsScale(<type(chemicalAbundances)> value) Set the scale of the chemicals property of the HotHaloStandard component.

<void> chemicalsSet(<type(chemicalAbundances)> value) Set the chemicals property of the hotHalo component.

<logical> coldModeIsActive() Return true if the coldMode implementation of the hotHalo component is the active choice.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a verySimpleDelayed implementation of the hotHalo component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a verySimpleDelayed implementation of the hotHalo component from array.

<void> destroy() Finalize a verySimpleDelayed implementation of the hotHalo component.

<void> dumpASCII() Dump the content of a hotHalo component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<double precision> heatSource() Returns the default value for the heatSource property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> heatSourceAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the heatSource property

<integer> heatSourceCount() Compute the count of evolvable quantities in the heatSource property of the HotHaloStandard component.

<logical> heatSourceIsGettable() Returns true if the heatSource property is gettable for the hotHalo component class.

<logical> heatSourceIsSettable() Specify whether the heatSource property of the hotHalo component is settable.

<void> heatSourceRate(<double precision> value) Cumulate to the rate of the heatSource property of the HotHaloStandard component.

<double precision> heatSourceRateGet() Returns a zero rate for the heatSource property for the hotHalo component class.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<type(abundances)> hotHaloCoolingAbundances() Returns the default value for the hotHaloCoolingAbundances property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> hotHaloCoolingAbundancesAttributeMatch(<l
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the hotHalo class that have the desired attributes for the hotHaloCoolingAbundances property

<integer> hotHaloCoolingAbundancesCount() Compute the count of evolvable quantities in the hotHaloCoolingAbundan
property of the HotHaloStandard component.

<logical> hotHaloCoolingAbundancesIsGettable() Returns true if the hotHaloCoolingAbundances property is gettable for the hotHalo component class.

<logical> hotHaloCoolingAbundancesIsSettable() Specify whether the hotHaloCoolingAbundances property of the hotHalo component is settable.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔,
<*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAbundancesRateFunction() Set the function to be used for the rate method of the hotHaloCoolingAbundances property of the hotHalo component.

<type(abundances)> hotHaloCoolingAbundancesRateGet() Returns a zero rate for the hotHaloCoolingAbundances property for the hotHalo component class.

<logical> hotHaloCoolingAbundancesRateIsAttached() Return true if the deferred function used to rate the hotHaloCoolingAbundances property of the hotHalo component class has been attached.

<double precision> hotHaloCoolingAngularMomentum() Returns the default value for the hotHaloCoolingAngularMome
property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> hotHaloCoolingAngularMomentumAttributeMat
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the hotHalo class that have the desired attributes for the hotHaloCoolingAngularMomentum property

<integer> hotHaloCoolingAngularMomentumCount() Compute the count of evolvable quantities in the hotHaloCoolingAngularMomentum property of the HotHaloStandard component.

<logical> hotHaloCoolingAngularMomentumIsGettable() Returns true if the hotHaloCoolingAngularMomentum property is gettable for the hotHalo component class.

<logical> hotHaloCoolingAngularMomentumIsSettable() Specify whether the hotHaloCoolingAngularMomentum property of the hotHalo component is settable.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRateFunction() Set the function to be used for the rate method of the hotHaloCoolingAngularMomentum property of the hotHalo component.

<double precision> hotHaloCoolingAngularMomentumRateGet() Returns a zero rate for the hotHaloCoolingAngularMo property for the hotHalo component class.

<logical> hotHaloCoolingAngularMomentumRateIsAttached() Return true if the deferred function used to rate the hotHaloCoolingAngularMomentum property of the hotHalo component class has been attached.

<double precision> hotHaloCoolingMass() Returns the default value for the hotHaloCoolingMass property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> hotHaloCoolingMassAttributeMatch(<logical [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the hotHaloCoolingMass property

<integer> hotHaloCoolingMassCount() Compute the count of evolvable quantities in the hotHaloCoolingMass property of the HotHaloStandard component.

<logical> hotHaloCoolingMassIsGettable() Returns true if the hotHaloCoolingMass property is gettable for the hotHalo component class.

<logical> hotHaloCoolingMassIsSettable() Specify whether the hotHaloCoolingMass property of the hotHalo component is settable.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRateFunction() Set the function to be used for the rate method of the hotHaloCoolingMass property of the hotHalo component.

<double precision> hotHaloCoolingMassRateGet() Returns a zero rate for the hotHaloCoolingMass property for the hotHalo component class.

<logical> hotHaloCoolingMassRateIsAttached() Return true if the deferred function used to rate the hotHaloCoolingMass property of the hotHalo component class has been attached.

<void> initialize() Initialize a verySimpleDelayed member of the hotHalo component.

<logical> isInitialized() Returns the default value for the isInitialized property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> isInitializedAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the isInitialized property

<logical> isInitializedIsGettable() Returns true if the isInitialized property is gettable for the hotHalo component class.

<logical> isInitializedIsSettable() Specify whether the isInitialized property of the hotHalo component is settable.

<logical> isInitializedRateGet() Returns a zero rate for the isInitialized property for the hotHalo component class.

<void> isInitializedSet(<logical> value) Set the isInitialized property of the hotHalo component.

<double precision> mass() Get the mass property of an verySimple implementation of the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> massAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the mass property

<double precision> massCold() Returns the default value for the massCold property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> massColdAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the massCold property

<integer> massColdCount() Compute the count of evolvable quantities in the massCold property of the HotHaloColdMode component.

<logical> massColdIsGettable() Returns true if the massCold property is gettable for the hotHalo component class.

<logical> massColdIsSettable() Specify whether the massCold property of the hotHalo component is settable.

<void> massColdRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accept a rate set for the massCold property of the hotHalo component class. Trigger an interrupt to create the component.

<double precision> massColdRateGet() Returns a zero rate for the massCold property for the hotHalo component class.

<void> massColdScale(<double precision> value) Set the scale of the massCold property of the HotHaloColdMode component.

<void> massColdSet(<double precision> value) Set the massCold property of the hotHalo component.

<integer> massCount() Return a count of the number of scalar properties in the mass property of an verySimple implementation of the hotHalo component class.

<void> massInactive() Indicate that the mass property of an verySimple implementation of the hotHalo component class is inactive for differential equation solving.

<logical> massIsGettable() Returns true if the mass property is gettable for the hotHalo component class.

<logical> massIsSettable() Specify whether the mass property of the hotHalo component is settable.

<void> massRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the mass property of an verySimple implementation of the hotHalo component class.

<double precision> massRateGet() Get the rate of change of the mass property of an verySimple implementation of the hotHalo component class.

<void> massScale(<double precision> setValue→) Set the absolute scale of the mass property of an verySimple implementation of the hotHalo component class.

<void> massSet(<double precision> setValue→) Set the mass property of an verySimple implementation of the hotHalo component class.

<double precision> massSink() Returns the default value for the massSink property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> massSinkAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the massSink property

<integer> massSinkCount() Compute the count of evolvable quantities in the massSink property of the HotHaloStandard component.

<logical> massSinkIsGettable() Returns true if the massSink property is gettable for the hotHalo component class.

<logical> massSinkIsSettable() Specify whether the massSink property of the hotHalo component is settable.

<void> massSinkRate(<double precision> value) Cumulate to the rate of the massSink property of the HotHaloStandard component.

<double precision> massSinkRateGet() Returns a zero rate for the massSink property for the hotHalo component class.

<double precision> massTotal() Returns the default value for the massTotal property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> massTotalAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the massTotal property

<logical> massTotalIsGettable() Returns true if the massTotal property is gettable for the hotHalo component class.

<logical> massTotalIsSettable() Specify whether the massTotal property of the hotHalo component is settable.

<double precision> massTotalRateGet() Returns a zero rate for the massTotal property for the hotHalo component class.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a verySimpleDelayed implementation of the hotHalo component class.

<logical> nullIsActive() Return true if the null implementation of the hotHalo component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<double precision> outerRadius() Get the value of the outerRadius property of the verySimple implementation of the hotHalo component using a deferred function.

<type(varying_string), allocatable, dimension(:) => matches> outerRadiusAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the outerRadius property

<integer> outerRadiusCount() Compute the count of evolvable quantities in the outerRadius property of the HotHaloStandard component.

<void> outerRadiusFunction() Set the function to be used for the get method of the outerRadius property of the HotHaloVerySimple component.

<logical> outerRadiusIsAttached() Return true if the deferred function used to get the outerRadius property of the HotHaloVerySimple component class has been attached.

<logical> outerRadiusIsGettable() Returns true if the outerRadius property is gettable for the hotHalo component class.

<logical> outerRadiusIsSettable() Specify whether the outerRadius property of the hotHalo component is settable.

<void> outerRadiusRate(<double precision> value) Cumulate to the rate of the outerRadius property of the HotHaloStandard component.

<double precision> outerRadiusRateGet() Returns a zero rate for the outerRadius property for the hotHalo component class.

<void> outerRadiusScale(<double precision> value) Set the scale of the outerRadius property of the HotHaloStandard component.

<void> outerRadiusSet(<double precision> value) Set the outerRadius property of the hotHalo component.

<type(abundances)> outflowedAbundances() Get the outflowedAbundances property of an verySimpleDelayed implementation of the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> outflowedAbundancesAttributeMatch(<logica
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the hotHalo class that have the desired attributes for the outflowedAbundances property

<integer> outflowedAbundancesCount() Return a count of the number of scalar properties in the outflowedAbundances property of an verySimpleDelayed implementation of the hotHalo component class.

<void> outflowedAbundancesInactive() Indicate that the outflowedAbundances property of an verySimpleDelayed implementation of the hotHalo component class is inactive for differential equation solving.

<logical> outflowedAbundancesIsGettable() Returns true if the outflowedAbundances property is gettable for the hotHalo component class.

<logical> outflowedAbundancesIsSettable() Specify whether the outflowedAbundances property of the hotHalo component is settable.

<void> outflowedAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔,
<*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the outflowedAbundances property of an verySimpleDelayed implementation of the hotHalo component class.

<type(abundances)> outflowedAbundancesRateGet() Get the rate of change of the outflowedAbundances property of an verySimpleDelayed implementation of the hotHalo component class.

<void> outflowedAbundancesScale(<type(abundances)> setValue→) Set the absolute scale of the outflowedAbundances property of an verySimpleDelayed implementation of the hotHalo component class.

<void> outflowedAbundancesSet(<type(abundances)> setValue→) Set the outflowedAbundances property of an verySimpleDelayed implementation of the hotHalo component class.

<double precision> outflowedAngularMomentum() Returns the default value for the outflowedAngularMomentum property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> outflowedAngularMomentumAttributeMatch(<
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the hotHalo class that have the desired attributes for the outflowedAngularMomentum property

<integer> outflowedAngularMomentumCount() Compute the count of evolvable quantities in the outflowedAngularMomen
property of the HotHaloStandard component.

<logical> outflowedAngularMomentumIsGettable() Returns true if the outflowedAngularMomentum property is gettable for the hotHalo component class.

<logical> outflowedAngularMomentumIsSettable() Specify whether the outflowedAngularMomentum property of the hotHalo component is settable.

<void> outflowedAngularMomentumRate(<double precision> value) Cumulate to the rate of the outflowedAngularMomentum property of the HotHaloStandard component.

<double precision> outflowedAngularMomentumRateGet() Returns a zero rate for the outflowedAngularMomentum property for the hotHalo component class.

<void> outflowedAngularMomentumScale(<double precision> value) Set the scale of the outflowedAngularMomentum property of the HotHaloStandard component.

<void> outflowedAngularMomentumSet(<double precision> value) Set the outflowedAngularMomentum property of the hotHalo component.

<double precision> outflowedMass() Get the outflowedMass property of an verySimpleDelayed implementation of the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> outflowedMassAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the outflowedMass property

<integer> outflowedMassCount() Return a count of the number of scalar properties in the outflowedMass property of an verySimpleDelayed implementation of the hotHalo component class.

<void> outflowedMassInactive() Indicate that the outflowedMass property of an verySimpleDelayed implementation of the hotHalo component class is inactive for differential equation solving.

<logical> outflowedMassIsGettable() Returns true if the outflowedMass property is gettable for the hotHalo component class.

<logical> outflowedMassIsSettable() Specify whether the outflowedMass property of the hotHalo component is settable.

<void> outflowedMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptT [interruptProcedure]↔) Accumulate to the rate of change of the outflowedMass property of an verySimpleDelayed implementation of the hotHalo component class.

<double precision> outflowedMassRateGet() Get the rate of change of the outflowedMass property of an verySimpleDelayed implementation of the hotHalo component class.

<void> outflowedMassScale(<double precision> setValue→) Set the absolute scale of the outflowedMass property of an verySimpleDelayed implementation of the hotHalo component class.

<void> outflowedMassSet(<double precision> setValue→) Set the outflowedMass property of an verySimpleDelayed implementation of the hotHalo component class.

<type(abundances)> outflowingAbundances() Returns the default value for the outflowingAbundances property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> outflowingAbundancesAttributeMatch(<logic [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the outflowingAbundances property

<integer> outflowingAbundancesCount() Compute the count of evolvable quantities in the outflowingAbundances property of the HotHaloStandard component.

<logical> outflowingAbundancesIsGettable() Returns true if the outflowingAbundances property is gettable for the hotHalo component class.

<logical> outflowingAbundancesIsSettable() Specify whether the outflowingAbundances property of the hotHalo component is settable.

<void> outflowingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the outflowingAbundances property of the verySimple implementation of the hotHalo component using a deferred function.

<void> outflowingAbundancesRateFunction() Set the function to be used for the rate method of the outflowingAbundances property of the HotHaloVerySimple component.

<type(abundances)> outflowingAbundancesRateGet() Returns a zero rate for the outflowingAbundances property for the hotHalo component class.

<logical> outflowingAbundancesRateIsAttached() Return true if the deferred function used to rate the outflowingAbundances property of the HotHaloVerySimple component class has been attached.

<double precision> outflowingAngularMomentum() Returns the default value for the outflowingAngularMomentum property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> outflowingAngularMomentumAttributeMatch(<
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the hotHalo class that have the desired attributes for the outflowingAngularMomentum property

<integer> outflowingAngularMomentumCount() Compute the count of evolvable quantities in the outflowingAngularMomentum property of the HotHaloStandard component.

<logical> outflowingAngularMomentumIsGettable() Returns true if the outflowingAngularMomentum property is gettable for the hotHalo component class.

<logical> outflowingAngularMomentumIsSettable() Specify whether the outflowingAngularMomentum property of the hotHalo component is settable.

<void> outflowingAngularMomentumRate(<double precision> value) Cumulate to the rate of the outflowingAngularMomentum property of the HotHaloStandard component.

<double precision> outflowingAngularMomentumRateGet() Returns a zero rate for the outflowingAngularMomentum property for the hotHalo component class.

<double precision> outflowingMass() Returns the default value for the outflowingMass property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> outflowingMassAttributeMatch(<logical>
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the hotHalo class that have the desired attributes for the outflowingMass property

<integer> outflowingMassCount() Compute the count of evolvable quantities in the outflowingMass property of the HotHaloStandard component.

<logical> outflowingMassIsGettable() Returns true if the outflowingMass property is gettable for the hotHalo component class.

<logical> outflowingMassIsSettable() Specify whether the outflowingMass property of the hotHalo component is settable.

<void> outflowingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interrupt [interruptProcedure]↔) Accumulate the rate of change of the outflowingMass property of the verySimple implementation of the hotHalo component using a deferred function.

<void> outflowingMassRateFunction() Set the function to be used for the rate method of the outflowingMass property of the HotHaloVerySimple component.

<double precision> outflowingMassRateGet() Returns a zero rate for the outflowingMass property for the hotHalo component class.

<logical> outflowingMassRateIsAttached() Return true if the deferred function used to rate the outflowingMass property of the HotHaloVerySimple component class has been attached.

<logical> outflowTrackingIsActive() Return true if the outflowTracking implementation of the hotHalo component is the active choice.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a hotHalo component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a verySimpleDelayed implementation of the hotHalo component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a verySimpleDelayed implementation of the hotHalo component.

<void> postOutput(<double precision> time→) Perform post-output processing for a verySimpleDelayed implementation of the hotHalo component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a verySimpleDelayed implementation of the hotHalo component.

<void> serializeASCII() Serialize the contents of a verySimpleDelayed implementation of the hotHalo component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a verySimpleDelayed implementation of the hotHalo component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a verySimpleDelayed implementation of the hotHalo component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a verySimpleDelayed implementation of the hotHalo component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a verySimpleDelayed implementation of the hotHalo component to XML.

<integer(c_size_t)> sizeOf() Return the size in bytes of a verySimpleDelayed implementation of the hotHalo component.

<logical> standardIsActive() Return true if the standard implementation of the hotHalo component is the active choice.

<type(abundances)> strippedAbundances() Returns the default value for the strippedAbundances property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> strippedAbundancesAttributeMatch(<logical [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the strippedAbundances property

<integer> strippedAbundancesCount() Compute the count of evolvable quantities in the strippedAbundances property of the HotHaloStandard component.

<logical> strippedAbundancesIsGettable() Returns true if the strippedAbundances property is gettable for the hotHalo component class.

<logical> strippedAbundancesIsSettable() Specify whether the strippedAbundances property of the hotHalo component is settable.

<void> strippedAbundancesRate(<type(abundances)> value) Cumulate to the rate of the strippedAbundances property of the HotHaloStandard component.

<type(abundances)> strippedAbundancesRateGet() Returns a zero rate for the strippedAbundances property for the hotHalo component class.

<void> strippedAbundancesScale(<type(abundances)> value) Set the scale of the strippedAbundances property of the HotHaloStandard component.

<void> strippedAbundancesSet(<type(abundances)> value) Set the strippedAbundances property of the hotHalo component.

<double precision> strippedMass() Returns the default value for the strippedMass property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> strippedMassAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the strippedMass property

<integer> strippedMassCount() Compute the count of evolvable quantities in the strippedMass property of the HotHaloStandard component.

<logical> strippedMassIsGettable() Returns true if the strippedMass property is gettable for the hotHalo component class.

<logical> strippedMassIsSettable() Specify whether the strippedMass property of the hotHalo component is settable.

<void> strippedMassRate(<double precision> value) Cumulate to the rate of the strippedMass property of the HotHaloStandard component.

<double precision> strippedMassRateGet() Returns a zero rate for the strippedMass property for the hotHalo component class.

<void> strippedMassScale(<double precision> value) Set the scale of the strippedMass property of the HotHaloStandard component.

<void> strippedMassSet(<double precision> value) Set the strippedMass property of the hotHalo component.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<type(abundances)> trackedOutflowAbundances() Returns the default value for the trackedOutflowAbundances property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> trackedOutflowAbundancesAttributeMatch(<l [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the trackedOutflowAbundances property

<integer> trackedOutflowAbundancesCount() Compute the count of evolvable quantities in the trackedOutflowAbundan property of the HotHaloOutflowTracking component.

<logical> trackedOutflowAbundancesIsGettable() Returns true if the trackedOutflowAbundances property is gettable for the hotHalo component class.

<logical> trackedOutflowAbundancesIsSettable() Specify whether the trackedOutflowAbundances property of the hotHalo component is settable.

<void> trackedOutflowAbundancesRate(<type(abundances)> value) Cumulate to the rate of the trackedOutflowAbundances property of the HotHaloOutflowTracking component.

<type(abundances)> trackedOutflowAbundancesRateGet() Returns a zero rate for the trackedOutflowAbundances property for the hotHalo component class.

<void> trackedOutflowAbundancesScale(<type(abundances)> value) Set the scale of the trackedOutflowAbundances property of the HotHaloOutflowTracking component.

<void> trackedOutflowAbundancesSet(<type(abundances)> value) Set the trackedOutflowAbundances property of the hotHalo component.

<double precision> trackedOutflowMass() Returns the default value for the trackedOutflowMass property for the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> trackedOutflowMassAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the trackedOutflowMass property

<integer> trackedOutflowMassCount() Compute the count of evolvable quantities in the trackedOutflowMass property of the HotHaloOutflowTracking component.

<logical> trackedOutflowMassIsGettable() Returns true if the trackedOutflowMass property is gettable for the hotHalo component class.

<logical> trackedOutflowMassIsSettable() Specify whether the trackedOutflowMass property of the hotHalo component is settable.

<void> trackedOutflowMassRate(<double precision> value) Cumulate to the rate of the trackedOutflowMass property of the HotHaloOutflowTracking component.

<double precision> trackedOutflowMassRateGet() Returns a zero rate for the trackedOutflowMass property for the hotHalo component class.

<void> trackedOutflowMassScale(<double precision> value) Set the scale of the trackedOutflowMass property of the HotHaloOutflowTracking component.

<void> trackedOutflowMassSet(<double precision> value) Set the trackedOutflowMass property of the hotHalo component.

<type(varying_string)> type() Returns the type name for the verySimpleDelayed implementation of the hotHalo component class.

<double precision> unaccretedMass() Get the unaccretedMass property of an verySimple implementation of the hotHalo component class.

<type(varying_string), allocatable, dimension(:) => matches> unaccretedMassAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the hotHalo class that have the desired attributes for the unaccretedMass property

<integer> unaccretedMassCount() Return a count of the number of scalar properties in the unaccretedMass property of an verySimple implementation of the hotHalo component class.

<void> unaccretedMassInactive() Indicate that the unaccretedMass property of an verySimple implementation of the hotHalo component class is inactive for differential equation solving.

<logical> unaccretedMassIsGettable() Returns true if the unaccretedMass property is gettable for the hotHalo component class.

<logical> unaccretedMassIsSettable() Specify whether the unaccretedMass property of the hotHalo component is settable.

<void> unaccretedMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interrupt [interruptProcedure]↔) Accumulate to the rate of change of the unaccretedMass property of an verySimple implementation of the hotHalo component class.

<double precision> unaccretedMassRateGet() Get the rate of change of the unaccretedMass property of an verySimple implementation of the hotHalo component class.

<void> unaccretedMassScale(<double precision> setValue→) Set the absolute scale of the unaccretedMass property of an verySimple implementation of the hotHalo component class.

<void> unaccretedMassSet(<double precision> setValue→) Set the unaccretedMass property of an verySimple implementation of the hotHalo component class.

<logical> verySimpleDelayedIsActive() Return true if the verySimpleDelayed implementation of the hotHalo component is the active choice.

<logical> verySimpleIsActive() Return true if the verySimple implementation of the hotHalo component is the active choice.

nodeComponentIndices

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<integer(kind=kind_int8) => classDefault> branchTip() Returns the default value for the branchTip property for the indices component class.

<type(varying_string), allocatable, dimension(:) => matches> branchTipAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the indices class that have the desired attributes for the branchTip property

<logical> branchTipIsGettable() Returns true if the branchTip property is gettable for the indices component class.

<logical> branchTipIsSettable() Specify whether the branchTip property of the indices component is settable.

<integer(kind=kind_int8) => classDefault> branchTipRateGet() Returns a zero rate for the branchTip property for the indices component class.

<void> branchTipSet(<integer(kind=kind_int8)> value) Set the branchTip property of the indices component.

<void> builder(<*type(node)> componentDefinition→) Build a generic indices component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Read properties from raw file.

<void> deserializeValues(<double(:)> array→, <integer> propertyType→) Deserialize the evolvable quantities from an array.

<void> destroy() Finalize a generic indices component.

<void> dumpASCII() Dump the content of a indices component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a generic indices component.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a nodeComponent object.

<logical> nullIsActive() Return true if the null implementation of the indices component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a indices component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a generic indices component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Establish the names of properties to output for a generic indices component.

<void> postOutput(<double precision> time→) Perform post-output processing of a indices component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets() Set offsets into serialization arrays.

<void> serializeASCII() Serialize the content of a indices component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the number of evolvable quantities to be evolved.

<void> serializeRaw(<integer> fileHandle→) Generate a binary dump of all properties.

<void> serializeValues(<double(:)> array←, <integer> propertyType→) Serialize the evolvable quantities to an array.

<void> serializeXML() Generate an XML dump of all properties.

<integer(c_size_t)> sizeOf() Return the size in bytes of a nodeComponentIndices component.

<logical> standardIsActive() Return true if the standard implementation of the indices component is the active choice.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<type(varying_string)> type() Returns the type name for the indices component class.

nodeComponentIndicesNull

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<integer(kind=kind_int8) => classDefault> branchTip() Returns the default value for the branchTip property for the indices component class.

<type(varying_string), allocatable, dimension(:) => matches> branchTipAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the indices class that have the desired attributes for the branchTip property

<logical> branchTipIsGettable() Returns true if the branchTip property is gettable for the indices component class.

<logical> branchTipIsSettable() Specify whether the branchTip property of the indices component is settable.

<integer(kind=kind_int8) => classDefault> branchTipRateGet() Returns a zero rate for the branchTip property for the indices component class.

<void> branchTipSet(<integer(kind=kind_int8)> value) Set the branchTip property of the indices component.

<void> builder(<*type(node)> componentDefinition→) Build a null implementation of the indices component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a null implementation of the indices component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a null implementation of the indices component from array.

<void> destroy() Finalize a null implementation of the indices component.

<void> dumpASCII() Dump the content of a indices component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a null member of the indices component.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a null implementation of the indices component class.

<logical> nullIsActive() Return true if the null implementation of the indices component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a indices component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a null implementation of the indices component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a null implementation of the indices component.

<void> postOutput(<double precision> time→) Perform post-output processing of a indices component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a null implementation of the indices component.

<void> serializeASCII() Serialize the contents of a null implementation of the indices component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a null implementation of the indices component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a null implementation of the indices component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a null implementation of the indices component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a null implementation of the indices component to XML.

<integer(c_size_t)> sizeOf() Return the size in bytes of a null implementation of the indices component.

<logical> standardIsActive() Return true if the standard implementation of the indices component is the active choice.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<type(varying_string)> type() Returns the type name for the null implementation of the indices component class.

nodeComponentIndicesStandard

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<integer(kind=kind_int8) => propertyValue> branchTip() Get the branchTip property of an standard implementation of the indices component class.

<type(varying_string), allocatable, dimension(:) => matches> branchTipAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the indices class that have the desired attributes for the branchTip property

<logical> branchTipIsGettable() Returns true if the branchTip property is gettable for the indices component class.

<logical> branchTipIsSettable() Specify whether the branchTip property of the indices component is settable.

<integer(kind=kind_int8) => classDefault> branchTipRateGet() Returns a zero rate for the branchTip property for the indices component class.

<void> branchTipSet(<integer(kind=kind_int8)> setValue→) Set the branchTip property of an standard implementation of the indices component class.

<void> builder(<*type(node)> componentDefinition→) Build a standard implementation of the indices component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a standard implementation of the indices component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a standard implementation of the indices component from array.

<void> destroy() Finalize a standard implementation of the indices component.

<void> dumpASCII() Dump the content of a indices component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a standard member of the indices component.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a standard implementation of the indices component class.

<logical> nullIsActive() Return true if the null implementation of the indices component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a indices component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a standard implementation of the indices component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a standard implementation of the indices component.

<void> postOutput(<double precision> time→) Perform post-output processing of a indices component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a standard implementation of the indices component.

<void> serializeASCII() Serialize the contents of a standard implementation of the indices component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a standard implementation of the indices component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a standard implementation of the indices component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a standard implementation of the indices component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a standard implementation of the indices component to XML.

<integer(c_size_t)> sizeOf() Return the size in bytes of a standard implementation of the indices component.

<logical> standardIsActive() Return true if the standard implementation of the indices component is the active choice.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<type(varying_string)> type() Returns the type name for the standard implementation of the indices component class.

nodeComponentInterOutput

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a generic interOutput component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Read properties from raw file.

<void> deserializeValues(<double(:)> array→, <integer> propertyType→) Deserialize the evolvable quantities from an array.

<void> destroy() Finalize a generic interOutput component.

<double precision> diskStarFormationRate() Returns the default value for the diskStarFormationRate property for the interOutput component class.

<type(varying_string), allocatable, dimension(:) => matches> diskStarFormationRateAttributeMatch(<logi [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the interOutput class that have the desired attributes for the diskStarFormationRate property

<integer> diskStarFormationRateCount() Compute the count of evolvable quantities in the diskStarFormationRate property of the InterOutputStandard component.

<logical> diskStarFormationRateIsGettable() Returns true if the diskStarFormationRate property is gettable for the interOutput component class.

<logical> diskStarFormationRateIsSettable() Specify whether the diskStarFormationRate property of the interOutput component is settable.

<void> diskStarFormationRateRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accept a rate set for the diskStarFormationRate property of the interOutput component class. Trigger an interrupt to create the component.

<double precision> diskStarFormationRateRateGet() Returns a zero rate for the diskStarFormationRate property for the interOutput component class.

<void> diskStarFormationRateScale(<double precision> value) Set the scale of the diskStarFormationRate property of the InterOutputStandard component.

<void> diskStarFormationRateSet(<double precision> value) Set the diskStarFormationRate property of the interOutput component.

<void> dumpASCII() Dump the content of a interOutput component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a generic interOutput component.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a nodeComponent object.

<logical> nullIsActive() Return true if the null implementation of the interOutput component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a interOutput component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a generic interOutput component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Establish the names of properties to output for a generic interOutput component.

<void> postOutput(<double precision> time→) Perform post-output processing of a interOutput component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets() Set offsets into serialization arrays.

<void> serializeASCII() Serialize the content of a interOutput component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the number of evolvable quantities to be evolved.

<void> serializeRaw(<integer> fileHandle→) Generate a binary dump of all properties.

<void> serializeValues(<double(:)> array←, <integer> propertyType→) Serialize the evolvable quantities to an array.

<void> serializeXML() Generate an XML dump of all properties.

<integer(c_size_t)> sizeOf() Return the size in bytes of a nodeComponentInterOutput component.

<double precision> spheroidStarFormationRate() Returns the default value for the spheroidStarFormationRate property for the interOutput component class.

<type(varying_string), allocatable, dimension(:) => matches> spheroidStarFormationRateAttributeMatch(< [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the interOutput class that have the desired attributes for the spheroidStarFormationRate property

<integer> spheroidStarFormationRateCount() Compute the count of evolvable quantities in the spheroidStarFormationRate property of the InterOutputStandard component.

<logical> spheroidStarFormationRateIsGettable() Returns true if the spheroidStarFormationRate property is gettable for the interOutput component class.

<logical> spheroidStarFormationRateIsSettable() Specify whether the spheroidStarFormationRate property of the interOutput component is settable.

<void> spheroidStarFormationRateRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accept a rate set for the spheroidStarFormationRate property of the interOutput component class. Trigger an interrupt to create the component.

<double precision> spheroidStarFormationRateRateGet() Returns a zero rate for the spheroidStarFormationRate property for the interOutput component class.

<void> spheroidStarFormationRateScale(<double precision> value) Set the scale of the spheroidStarFormationRat property of the InterOutputStandard component.

<void> spheroidStarFormationRateSet(<double precision> value) Set the spheroidStarFormationRate property of the interOutput component.

<logical> standardIsActive() Return true if the standard implementation of the interOutput component is the active choice.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<type(varying_string)> type() Returns the type name for the interOutput component class.

nodeComponentInterOutputNull

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a null implementation of the interOutput component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a null implementation of the interOutput component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a null implementation of the interOutput component from array.

<void> destroy() Finalize a null implementation of the interOutput component.

<double precision> diskStarFormationRate() Returns the default value for the diskStarFormationRate property for the interOutput component class.

<type(varying_string), allocatable, dimension(:) => matches> diskStarFormationRateAttributeMatch(<logi [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the interOutput class that have the desired attributes for the diskStarFormationRate property

<integer> diskStarFormationRateCount() Compute the count of evolvable quantities in the diskStarFormationRate property of the InterOutputStandard component.

<logical> diskStarFormationRateIsGettable() Returns true if the diskStarFormationRate property is gettable for the interOutput component class.

<logical> diskStarFormationRateIsSettable() Specify whether the diskStarFormationRate property of the interOutput component is settable.

<void> diskStarFormationRateRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accept a rate set for the diskStarFormationRate property of the interOutput component class. Trigger an interrupt to create the component.

<double precision> diskStarFormationRateRateGet() Returns a zero rate for the diskStarFormationRate property for the interOutput component class.

<void> diskStarFormationRateScale(<double precision> value) Set the scale of the diskStarFormationRate property of the InterOutputStandard component.

<void> diskStarFormationRateSet(<double precision> value) Set the diskStarFormationRate property of the interOutput component.

<void> dumpASCII() Dump the content of a interOutput component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a null member of the interOutput component.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a null implementation of the interOutput component class.

<logical> nullIsActive() Return true if the null implementation of the interOutput component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a interOutput component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a null implementation of the interOutput component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a null implementation of the interOutput component.

<void> postOutput(<double precision> time→) Perform post-output processing of a interOutput component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a null implementation of the interOutput component.

<void> serializeASCII() Serialize the contents of a null implementation of the interOutput component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a null implementation of the interOutput component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a null implementation of the interOutput component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a null implementation of the interOutput component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a null implementation of the interOutput component to XML.

<integer(c_size_t)> sizeOf() Return the size in bytes of a null implementation of the interOutput component.

<double precision> spheroidStarFormationRate() Returns the default value for the spheroidStarFormationRate property for the interOutput component class.

<type(varying_string), allocatable, dimension(:) => matches> spheroidStarFormationRateAttributeMatch(<[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the interOutput class that have the desired attributes for the spheroidStarFormationRate property

<integer> spheroidStarFormationRateCount() Compute the count of evolvable quantities in the spheroidStarFormationRate property of the InterOutputStandard component.

<logical> spheroidStarFormationRateIsGettable() Returns true if the spheroidStarFormationRate property is gettable for the interOutput component class.

<logical> spheroidStarFormationRateIsSettable() Specify whether the spheroidStarFormationRate property of the interOutput component is settable.

<void> spheroidStarFormationRateRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accept a rate set for the spheroidStarFormationRate property of the interOutput component class. Trigger an interrupt to create the component.

<double precision> spheroidStarFormationRateRateGet() Returns a zero rate for the spheroidStarFormationRate property for the interOutput component class.

<void> spheroidStarFormationRateScale(<double precision> value) Set the scale of the spheroidStarFormationRat property of the InterOutputStandard component.

<void> spheroidStarFormationRateSet(<double precision> value) Set the spheroidStarFormationRate property of the interOutput component.

<logical> standardIsActive() Return true if the standard implementation of the interOutput component is the active choice.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<type(varying_string)> type() Returns the type name for the null implementation of the interOutput component class.

nodeComponentInterOutputStandard

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a standard implementation of the interOutput component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a standard implementation of the interOutput component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a standard implementation of the interOutput component from array.

<void> destroy() Finalize a standard implementation of the interOutput component.

<double precision> diskStarFormationRate() Get the diskStarFormationRate property of an standard implementation of the interOutput component class.

<type(varying_string), allocatable, dimension(:) => matches> diskStarFormationRateAttributeMatch(<logi [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the interOutput class that have the desired attributes for the diskStarFormationRate property

<integer> diskStarFormationRateCount() Return a count of the number of scalar properties in the diskStarFormationRate property of an standard implementation of the interOutput component class.

<void> diskStarFormationRateInactive() Indicate that the diskStarFormationRate property of an standard implementation of the interOutput component class is inactive for differential equation solving.

<logical> diskStarFormationRateIsGettable() Returns true if the diskStarFormationRate property is gettable for the interOutput component class.

<logical> diskStarFormationRateIsSettable() Specify whether the diskStarFormationRate property of the interOutput component is settable.

<void> diskStarFormationRateRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the diskStarFormationRate property of an standard implementation of the interOutput component class.

<double precision> diskStarFormationRateRateGet() Get the rate of change of the diskStarFormationRate property of an standard implementation of the interOutput component class.

<void> diskStarFormationRateScale(<double precision> setValue→) Set the absolute scale of the diskStarFormationRate property of an standard implementation of the interOutput component class.

<void> diskStarFormationRateSet(<double precision> setValue→) Set the diskStarFormationRate property of an standard implementation of the interOutput component class.

<void> dumpASCII() Dump the content of a interOutput component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a standard member of the interOutput component.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a standard implementation of the interOutput component class.

<logical> nullIsActive() Return true if the null implementation of the interOutput component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a interOutput component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a standard implementation of the interOutput component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a standard implementation of the interOutput component.

<void> postOutput(<double precision> time→) Perform post-output processing of a interOutput component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a standard implementation of the interOutput component.

<void> serializeASCII() Serialize the contents of a standard implementation of the interOutput component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a standard implementation of the interOutput component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a standard implementation of the interOutput component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a standard implementation of the interOutput component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a standard implementation of the interOutput component to XML.

<integer(c_size_t)> sizeOf() Return the size in bytes of a standard implementation of the interOutput component.

<double precision> spheroidStarFormationRate() Get the spheroidStarFormationRate property of an standard implementation of the interOutput component class.

<type(varying_string), allocatable, dimension(:) => matches> spheroidStarFormationRateAttributeMatch([requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the interOutput class that have the desired attributes for the spheroidStarFormationRate property

<integer> spheroidStarFormationRateCount() Return a count of the number of scalar properties in the spheroidStarFormationRate property of an standard implementation of the interOutput component class.

<void> spheroidStarFormationRateInactive() Indicate that the spheroidStarFormationRate property of an standard implementation of the interOutput component class is inactive for differential equation solving.

<logical> spheroidStarFormationRateIsGettable() Returns true if the spheroidStarFormationRate property is gettable for the interOutput component class.

<logical> spheroidStarFormationRateIsSettable() Specify whether the spheroidStarFormationRate property of the interOutput component is settable.

<void> spheroidStarFormationRateRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the spheroidStarFormationRate property of an standard implementation of the interOutput component class.

<double precision> spheroidStarFormationRateRateGet() Get the rate of change of the spheroidStarFormationRate property of an standard implementation of the interOutput component class.

<void> spheroidStarFormationRateScale(<double precision> setValue→) Set the absolute scale of the spheroidStarFormationRate property of an standard implementation of the interOutput component class.

<void> spheroidStarFormationRateSet(<double precision> setValue→) Set the spheroidStarFormationRate property of an standard implementation of the interOutput component class.

<logical> standardIsActive() Return true if the standard implementation of the interOutput component is the active choice.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<type(varying_string)> type() Returns the type name for the standard implementation of the interOutput component class.

nodeComponentMassFlowStatistics

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a generic massFlowStatistics component from a supplied XML definition.

<double precision> cooledMass() Returns the default value for the cooledMass property for the massFlowStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> cooledMassAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the massFlowStatistics class that have the desired attributes for the cooledMass property

<integer> cooledMassCount() Compute the count of evolvable quantities in the cooledMass property of the MassFlowStatisticsStandard component.

<logical> cooledMassIsGettable() Returns true if the cooledMass property is gettable for the massFlowStatistics component class.

<logical> cooledMassIsSettable() Specify whether the cooledMass property of the massFlowStatistics component is settable.

<void> cooledMassRate(<double precision> value) Cumulate to the rate of the cooledMass property of the MassFlowStatisticsStandard component.

<double precision> cooledMassRateGet() Returns a zero rate for the cooledMass property for the massFlowStatistics component class.

<void> cooledMassScale(<double precision> value) Set the scale of the cooledMass property of the MassFlowStatisticsStandard component.

<void> cooledMassSet(<double precision> value) Set the cooledMass property of the massFlowStatistics component.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Read properties from raw file.

<void> deserializeValues(<double(:)> array→, <integer> propertyType→) Deserialize the evolvable quantities from an array.

<void> destroy() Finalize a generic massFlowStatistics component.

<void> dumpASCII() Dump the content of a massFlowStatistics component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a generic massFlowStatistics component.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a nodeComponent object.

<logical> nullIsActive() Return true if the null implementation of the massFlowStatistics component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a massFlowStatistics component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a generic massFlowStatistics component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Establish the names of properties to output for a generic massFlowStatistics component.

<void> postOutput(<double precision> time→) Perform post-output processing of a massFlowStatistics component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets() Set offsets into serialization arrays.

<void> serializeASCII() Serialize the content of a massFlowStatistics component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the number of evolvable quantities to be evolved.

<void> serializeRaw(<integer> fileHandle→) Generate a binary dump of all properties.

<void> serializeValues(<double(:)> array←, <integer> propertyType→) Serialize the evolvable quantities to an array.

<void> serializeXML() Generate an XML dump of all properties.

<integer(c_size_t)> sizeOf() Return the size in bytes of a nodeComponentMassFlowStatistics component.

<logical> standardIsActive() Return true if the standard implementation of the massFlowStatistics component is the active choice.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<type(varying_string)> type() Returns the type name for the massFlowStatistics component class.

nodeComponentMassFlowStatisticsNull

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a null implementation of the massFlowStatistics component from a supplied XML definition.

<double precision> cooledMass() Returns the default value for the cooledMass property for the massFlowStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> cooledMassAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the massFlowStatistics class that have the desired attributes for the cooledMass property

<integer> cooledMassCount() Compute the count of evolvable quantities in the cooledMass property of the MassFlowStatisticsStandard component.

<logical> cooledMassIsGettable() Returns true if the cooledMass property is gettable for the massFlowStatistics component class.

<logical> cooledMassIsSettable() Specify whether the cooledMass property of the massFlowStatistics component is settable.

<void> cooledMassRate(<double precision> value) Cumulate to the rate of the cooledMass property of the MassFlowStatisticsStandard component.

<double precision> cooledMassRateGet() Returns a zero rate for the cooledMass property for the massFlowStatistics component class.

<void> cooledMassScale(<double precision> value) Set the scale of the cooledMass property of the MassFlowStatisticsStandard component.

<void> cooledMassSet(<double precision> value) Set the cooledMass property of the massFlowStatistics component.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a null implementation of the massFlowStatistics component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a null implementation of the massFlowStatistics component from array.

<void> destroy() Finalize a null implementation of the massFlowStatistics component.

<void> dumpASCII() Dump the content of a massFlowStatistics component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a null member of the massFlowStatistics component.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a null implementation of the massFlowStatistics component class.

<logical> nullIsActive() Return true if the null implementation of the massFlowStatistics component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_-int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a massFlowStatistics component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a null implementation of the massFlowStatistics component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a null implementation of the massFlowStatistics component.

<void> postOutput(<double precision> time→) Perform post-output processing of a massFlowStatistics component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a null implementation of the massFlowStatistics component.

<void> serializeASCII() Serialize the contents of a null implementation of the massFlowStatistics component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a null implementation of the massFlowStatistics component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a null implementation of the massFlowStatistics component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a null implementation of the massFlowStatistics component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a null implementation of the massFlowStatistics component to XML.

<integer(c_size_t)> sizeOf() Return the size in bytes of a null implementation of the massFlowStatistics component.

<logical> standardIsActive() Return true if the standard implementation of the massFlowStatistics component is the active choice.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<type(varying_string)> type() Returns the type name for the null implementation of the massFlowStatistics component class.

nodeComponentMassFlowStatisticsStandard

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a standard implementation of the massFlowStatistics component from a supplied XML definition.

<double precision> cooledMass() Get the cooledMass property of an standard implementation of the massFlowStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> cooledMassAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the massFlowStatistics class that have the desired attributes for the cooledMass property

<integer> cooledMassCount() Return a count of the number of scalar properties in the cooledMass property of an standard implementation of the massFlowStatistics component class.

<void> cooledMassInactive() Indicate that the cooledMass property of an standard implementation of the massFlowStatistics component class is inactive for differential equation solving.

<logical> cooledMassIsGettable() Returns true if the cooledMass property is gettable for the massFlowStatistics component class.

<logical> cooledMassIsSettable() Specify whether the cooledMass property of the massFlowStatistics component is settable.

<void> cooledMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask [interruptProcedure]↔) Accumulate to the rate of change of the cooledMass property of an standard implementation of the massFlowStatistics component class.

<double precision> cooledMassRateGet() Get the rate of change of the cooledMass property of an standard implementation of the massFlowStatistics component class.

<void> cooledMassScale(<double precision> setValue→) Set the absolute scale of the cooledMass property of an standard implementation of the massFlowStatistics component class.

<void> cooledMassSet(<double precision> setValue→) Set the cooledMass property of an standard implementation of the massFlowStatistics component class.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a standard implementation of the massFlowStatistics component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a standard implementation of the massFlowStatistics component from array.

<void> destroy() Finalize a standard implementation of the massFlowStatistics component.

<void> dumpASCII() Dump the content of a massFlowStatistics component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter
[interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a standard member of the massFlowStatistics component.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a standard implementation of the massFlowStatistics component class.

<logical> nullIsActive() Return true if the null implementation of the massFlowStatistics component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a massFlowStatistics component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a standard implementation of the massFlowStatistics component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision>

time→, <integer> instance→) Return the names of properties to output for a standard implementation of the massFlowStatistics component.

<void> postOutput(<double precision> time→) Perform post-output processing of a massFlowStatistics component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a standard implementation of the massFlowStatistics component.

<void> serializeASCII() Serialize the contents of a standard implementation of the massFlowStatistics component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a standard implementation of the massFlowStatistics component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a standard implementation of the massFlowStatistics component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a standard implementation of the massFlowStatistics component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a standard implementation of the massFlowStatistics component to XML.

<integer(c_size_t)> sizeOf() Return the size in bytes of a standard implementation of the massFlowStatistics component.

<logical> standardIsActive() Return true if the standard implementation of the massFlowStatistics component is the active choice.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<type(varying_string)> type() Returns the type name for the standard implementation of the massFlowStatistics component class.

nodeComponentMergingStatistics

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a generic mergingStatistics component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Read properties from raw file.

<void> deserializeValues(<double(:)> array→, <integer> propertyType→) Deserialize the evolvable quantities from an array.

<void> destroy() Finalize a generic mergingStatistics component.

<void> dumpASCII() Dump the content of a mergingStatistics component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<double precision> galaxyMajorMergerTime() Returns the default value for the galaxyMajorMergerTime property for the mergingStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> galaxyMajorMergerTimeAttributeMatch(<logi [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the mergingStatistics class that have the desired attributes for the galaxyMajorMergerTime property

<logical> galaxyMajorMergerTimeIsGettable() Returns true if the galaxyMajorMergerTime property is gettable for the mergingStatistics component class.

<logical> galaxyMajorMergerTimeIsSettable() Specify whether the galaxyMajorMergerTime property of the mergingStatistics component is settable.

<double precision> galaxyMajorMergerTimeRateGet() Returns a zero rate for the galaxyMajorMergerTime property for the mergingStatistics component class.

<void> galaxyMajorMergerTimeSet(<double precision> value) Set the galaxyMajorMergerTime property of the mergingStatistics component.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a generic mergingStatistics component.

<logical> majorIsActive() Return true if the major implementation of the mergingStatistics component is the active choice.

<double precision, dimension(:), allocatable => classDefault> majorMergerTime() Returns the default value for the majorMergerTime property for the mergingStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> majorMergerTimeAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the mergingStatistics class that have the desired attributes for the majorMergerTime property

<logical> majorMergerTimeIsGettable() Returns true if the majorMergerTime property is gettable for the mergingStatistics component class.

<logical> majorMergerTimeIsSettable() Specify whether the majorMergerTime property of the mergingStatistics component is settable.

<double precision, dimension(:), allocatable => classDefault> majorMergerTimeRateGet() Returns a zero rate for the majorMergerTime property for the mergingStatistics component class.

<void> majorMergerTimeSet(<double precision[:]> value) Set the majorMergerTime property of the mergingStatistics component.

<double precision> massWhenFirstIsolated() Returns the default value for the massWhenFirstIsolated property for the mergingStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> massWhenFirstIsolatedAttributeMatch(<logi [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the mergingStatistics class that have the desired attributes for the massWhenFirstIsolated property

<logical> massWhenFirstIsolatedIsGettable() Returns true if the massWhenFirstIsolated property is gettable for the mergingStatistics component class.

<logical> massWhenFirstIsolatedIsSettable() Specify whether the massWhenFirstIsolated property of the mergingStatistics component is settable.

<double precision> massWhenFirstIsolatedRateGet() Returns a zero rate for the massWhenFirstIsolated property for the mergingStatistics component class.

<void> massWhenFirstIsolatedSet(<double precision> value) Set the massWhenFirstIsolated property of the mergingStatistics component.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a nodeComponent object.

<double precision> nodeFormationTime() Returns the default value for the nodeFormationTime property for the mergingStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> nodeFormationTimeAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the mergingStatistics class that have the desired attributes for the nodeFormationTime property

<logical> nodeFormationTimeIsGettable() Returns true if the nodeFormationTime property is gettable for the mergingStatistics component class.

<logical> nodeFormationTimeIsSettable() Specify whether the nodeFormationTime property of the mergingStatistics component is settable.

<double precision> nodeFormationTimeRateGet() Returns a zero rate for the nodeFormationTime property for the mergingStatistics component class.

<void> nodeFormationTimeSet(<double precision> value) Set the nodeFormationTime property of the mergingStatistics component.

<integer> nodeHierarchyLevel() Returns the default value for the nodeHierarchyLevel property for the mergingStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> nodeHierarchyLevelAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the mergingStatistics class that have the desired attributes for the nodeHierarchyLevel property

<logical> nodeHierarchyLevelIsGettable() Returns true if the nodeHierarchyLevel property is gettable for the mergingStatistics component class.

<logical> nodeHierarchyLevelIsSettable() Specify whether the nodeHierarchyLevel property of the mergingStatistics component is settable.

<integer> nodeHierarchyLevelMaximum() Returns the default value for the nodeHierarchyLevelMaximum property for the mergingStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> nodeHierarchyLevelMaximumAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the mergingStatistics class that have the desired attributes for the nodeHierarchyLevelMaximum property

<logical> nodeHierarchyLevelMaximumIsGettable() Returns true if the nodeHierarchyLevelMaximum property is gettable for the mergingStatistics component class.

<logical> nodeHierarchyLevelMaximumIsSettable() Specify whether the nodeHierarchyLevelMaximum property of the mergingStatistics component is settable.

<integer> nodeHierarchyLevelMaximumRateGet() Returns a zero rate for the nodeHierarchyLevelMaximum property for the mergingStatistics component class.

<void> nodeHierarchyLevelMaximumSet(<integer> value) Set the nodeHierarchyLevelMaximum property of the mergingStatistics component.

<integer> nodeHierarchyLevelRateGet() Returns a zero rate for the nodeHierarchyLevel property for the mergingStatistics component class.

<void> nodeHierarchyLevelSet(<integer> value) Set the nodeHierarchyLevel property of the mergingStatistics component.

<double precision> nodeMajorMergerTime() Returns the default value for the nodeMajorMergerTime property for the mergingStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> nodeMajorMergerTimeAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the mergingStatistics class that have the desired attributes for the nodeMajorMergerTime property

<logical> nodeMajorMergerTimeIsGettable() Returns true if the nodeMajorMergerTime property is gettable for the mergingStatistics component class.

<logical> nodeMajorMergerTimeIsSettable() Specify whether the nodeMajorMergerTime property of the mergingStatistics component is settable.

<double precision> nodeMajorMergerTimeRateGet() Returns a zero rate for the nodeMajorMergerTime property for the mergingStatistics component class.

<void> nodeMajorMergerTimeSet(<double precision> value) Set the nodeMajorMergerTime property of the mergingStatistics component.

<logical> nullIsActive() Return true if the null implementation of the mergingStatistics component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a mergingStatistics component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a generic mergingStatistics component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Establish the names of properties to output for a generic mergingStatistics component.

<void> postOutput(<double precision> time→) Perform post-output processing of a mergingStatistics component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<logical> recentIsActive() Return true if the recent implementation of the mergingStatistics component is the active choice.

<integer, dimension(:), allocatable => classDefault> recentMajorMergerCount() Returns the default value for the recentMajorMergerCount property for the mergingStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> recentMajorMergerCountAttributeMatch(<log [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the mergingStatistics class that have the desired attributes for the recentMajorMergerCount property

<logical> recentMajorMergerCountIsGettable() Returns true if the recentMajorMergerCount property is gettable for the mergingStatistics component class.

<logical> recentMajorMergerCountIsSettable() Specify whether the recentMajorMergerCount property of the mergingStatistics component is settable.

<integer, dimension(:), allocatable => classDefault> recentMajorMergerCountRateGet() Returns a zero rate for the recentMajorMergerCount property for the mergingStatistics component class.

<void> recentMajorMergerCountSet(<integer[:]> value) Set the recentMajorMergerCount property of the mergingStatistics component.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets() Set offsets into serialization arrays.

<void> serializeASCII() Serialize the content of a mergingStatistics component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the number of evolvable quantities to be evolved.

<void> serializeRaw(<integer> fileHandle→) Generate a binary dump of all properties.

<void> serializeValues(<double(:)> array←, <integer> propertyType→) Serialize the evolvable quantities to an array.

<void> serializeXML() Generate an XML dump of all properties.

<integer(c_size_t)> sizeOf() Return the size in bytes of a nodeComponentMergingStatistics component.

<logical> standardIsActive() Return true if the standard implementation of the mergingStatistics component is the active choice.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<type(varying_string)> type() Returns the type name for the mergingStatistics component class.

nodeComponentMergingStatisticsMajor

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a major implementation of the mergingStatistics component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a major implementation of the mergingStatistics component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a major implementation of the mergingStatistics component from array.

<void> destroy() Finalize a major implementation of the mergingStatistics component.

<void> dumpASCII() Dump the content of a mergingStatistics component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<double precision> galaxyMajorMergerTime() Returns the default value for the galaxyMajorMergerTime property for the mergingStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> galaxyMajorMergerTimeAttributeMatch(<logi [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the mergingStatistics class that have the desired attributes for the galaxyMajorMergerTime property

<logical> galaxyMajorMergerTimeIsGettable() Returns true if the galaxyMajorMergerTime property is gettable for the mergingStatistics component class.

<logical> galaxyMajorMergerTimeIsSettable() Specify whether the galaxyMajorMergerTime property of the mergingStatistics component is settable.

<double precision> galaxyMajorMergerTimeRateGet() Returns a zero rate for the galaxyMajorMergerTime property for the mergingStatistics component class.

<void> galaxyMajorMergerTimeSet(<double precision> value) Set the galaxyMajorMergerTime property of the mergingStatistics component.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a major member of the mergingStatistics component.

<logical> majorIsActive() Return true if the major implementation of the mergingStatistics component is the active choice.

<double precision, dimension(:), allocatable => propertyValue> majorMergerTime() Get the majorMergerTime property of an major implementation of the mergingStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> majorMergerTimeAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the mergingStatistics class that have the desired attributes for the majorMergerTime property

<logical> majorMergerTimeIsGettable() Returns true if the majorMergerTime property is gettable for the mergingStatistics component class.

<logical> majorMergerTimeIsSettable() Specify whether the majorMergerTime property of the mergingStatistics component is settable.

<double precision, dimension(:), allocatable => classDefault> majorMergerTimeRateGet()
Returns a zero rate for the majorMergerTime property for the mergingStatistics component class.

<void> majorMergerTimeSet(<double precision[:]> setValue→) Set the majorMergerTime property of an major implementation of the mergingStatistics component class.

<double precision> massWhenFirstIsolated() Returns the default value for the massWhenFirstIsolated property for the mergingStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> massWhenFirstIsolatedAttributeMatch(<logi
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the mergingStatistics class that have the desired attributes for the massWhenFirstIsolated property

<logical> massWhenFirstIsolatedIsGettable() Returns true if the massWhenFirstIsolated property is gettable for the mergingStatistics component class.

<logical> massWhenFirstIsolatedIsSettable() Specify whether the massWhenFirstIsolated property of the mergingStatistics component is settable.

<double precision> massWhenFirstIsolatedRateGet() Returns a zero rate for the massWhenFirstIsolated property for the mergingStatistics component class.

<void> massWhenFirstIsolatedSet(<double precision> value) Set the massWhenFirstIsolated property of the mergingStatistics component.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a major implementation of the mergingStatistics component class.

<double precision> nodeFormationTime() Returns the default value for the nodeFormationTime property for the mergingStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> nodeFormationTimeAttributeMatch(<logical>
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the mergingStatistics class that have the desired attributes for the nodeFormationTime property

<logical> nodeFormationTimeIsGettable() Returns true if the nodeFormationTime property is gettable for the mergingStatistics component class.

<logical> nodeFormationTimeIsSettable() Specify whether the nodeFormationTime property of the mergingStatistics component is settable.

<double precision> nodeFormationTimeRateGet() Returns a zero rate for the nodeFormationTime property for the mergingStatistics component class.

<void> nodeFormationTimeSet(<double precision> value) Set the nodeFormationTime property of the mergingStatistics component.

<integer> nodeHierarchyLevel() Returns the default value for the nodeHierarchyLevel property for the mergingStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> nodeHierarchyLevelAttributeMatch(<logical [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the mergingStatistics class that have the desired attributes for the nodeHierarchyLevel property

<logical> nodeHierarchyLevelIsGettable() Returns true if the nodeHierarchyLevel property is gettable for the mergingStatistics component class.

<logical> nodeHierarchyLevelIsSettable() Specify whether the nodeHierarchyLevel property of the mergingStatistics component is settable.

<integer> nodeHierarchyLevelMaximum() Returns the default value for the nodeHierarchyLevelMaximum property for the mergingStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> nodeHierarchyLevelMaximumAttributeMatch(< [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the mergingStatistics class that have the desired attributes for the nodeHierarchyLevelMaximum property

<logical> nodeHierarchyLevelMaximumIsGettable() Returns true if the nodeHierarchyLevelMaximum property is gettable for the mergingStatistics component class.

<logical> nodeHierarchyLevelMaximumIsSettable() Specify whether the nodeHierarchyLevelMaximum property of the mergingStatistics component is settable.

<integer> nodeHierarchyLevelMaximumRateGet() Returns a zero rate for the nodeHierarchyLevelMaximum property for the mergingStatistics component class.

<void> nodeHierarchyLevelMaximumSet(<integer> value) Set the nodeHierarchyLevelMaximum property of the mergingStatistics component.

<integer> nodeHierarchyLevelRateGet() Returns a zero rate for the nodeHierarchyLevel property for the mergingStatistics component class.

<void> nodeHierarchyLevelSet(<integer> value) Set the nodeHierarchyLevel property of the mergingStatistics component.

<double precision> nodeMajorMergerTime() Returns the default value for the nodeMajorMergerTime property for the mergingStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> nodeMajorMergerTimeAttributeMatch(<logica [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the mergingStatistics class that have the desired attributes for the nodeMajorMergerTime property

<logical> nodeMajorMergerTimeIsGettable() Returns true if the nodeMajorMergerTime property is gettable for the mergingStatistics component class.

<logical> nodeMajorMergerTimeIsSettable() Specify whether the nodeMajorMergerTime property of the mergingStatistics component is settable.

<double precision> nodeMajorMergerTimeRateGet() Returns a zero rate for the nodeMajorMergerTime property for the mergingStatistics component class.

<void> nodeMajorMergerTimeSet(<double precision> value) Set the nodeMajorMergerTime property of the mergingStatistics component.

<logical> nullIsActive() Return true if the null implementation of the mergingStatistics component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_-int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a mergingStatistics component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a major implementation of the mergingStatistics component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a major implementation of the mergingStatistics component.

<void> postOutput(<double precision> time→) Perform post-output processing of a mergingStatistics component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<logical> recentIsActive() Return true if the recent implementation of the mergingStatistics component is the active choice.

<integer, dimension(:), allocatable => classDefault> recentMajorMergerCount() Returns the default value for the recentMajorMergerCount property for the mergingStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> recentMajorMergerCountAttributeMatch(<log [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the mergingStatistics class that have the desired attributes for the recentMajorMergerCount property

<logical> recentMajorMergerCountIsGettable() Returns true if the recentMajorMergerCount property is gettable for the mergingStatistics component class.

`<logical> recentMajorMergerCountIsSettable()` Specify whether the `recentMajorMergerCount` property of the `mergingStatistics` component is settable.

`<integer, dimension(:), allocatable => classDefault> recentMajorMergerCountRateGet()` Returns a zero rate for the `recentMajorMergerCount` property for the `mergingStatistics` component class.

`<void> recentMajorMergerCountSet(<integer[:]> value)` Set the `recentMajorMergerCount` property of the `mergingStatistics` component.

`<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→)` Compute the rotation curve.

`<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→)` Compute the rotation curve gradient.

`<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→)` Compute offsets into serialization arrays for a `major` implementation of the `mergingStatistics` component.

`<void> serializeASCII()` Serialize the contents of a major implementation of the mergingStatistics component to ASCII.

`<integer> serializeCount(<integer> propertyType→)` Return a count of the serialization of a major implementation of the mergingStatistics component.

`<void> serializeRaw(<integer> fileHandle→)` Serialize the contents of a major implementation of the mergingStatistics component to raw (binary) file.

`<void> serializeValues(<double precision[:]> array←, <integer> propertyType→)` Serialize evolvable properties of a `major` implementation of the `mergingStatistics` component to array.

`<void> serializeXML(<integer> fileHandle→)` Serialize the contents of a major implementation of the mergingStatistics component to XML.

`<integer(c_size_t)> sizeOf()` Return the size in bytes of a major implementation of the mergingStatistics component.

`<logical> standardIsActive()` Return true if the standard implementation of the mergingStatistics component is the active choice.

`<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→)` Compute the surface density.

`<type(varying_string)> type()` Returns the type name for the major implementation of the mergingStatistics component class.

nodeComponentMergingStatisticsNull

`<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→)` Assign a node component to another node component.

`<void> builder(<*type(node)> componentDefinition→)` Build a `null` implementation of the `mergingStatistics` component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a null implementation of the mergingStatistics component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a null implementation of the mergingStatistics component from array.

<void> destroy() Finalize a null implementation of the mergingStatistics component.

<void> dumpASCII() Dump the content of a mergingStatistics component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<double precision> galaxyMajorMergerTime() Returns the default value for the galaxyMajorMergerTime property for the mergingStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> galaxyMajorMergerTimeAttributeMatch(<logi [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the mergingStatistics class that have the desired attributes for the galaxyMajorMergerTime property

<logical> galaxyMajorMergerTimeIsGettable() Returns true if the galaxyMajorMergerTime property is gettable for the mergingStatistics component class.

<logical> galaxyMajorMergerTimeIsSettable() Specify whether the galaxyMajorMergerTime property of the mergingStatistics component is settable.

<double precision> galaxyMajorMergerTimeRateGet() Returns a zero rate for the galaxyMajorMergerTime property for the mergingStatistics component class.

<void> galaxyMajorMergerTimeSet(<double precision> value) Set the galaxyMajorMergerTime property of the mergingStatistics component.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a null member of the mergingStatistics component.

<logical> majorIsActive() Return true if the major implementation of the mergingStatistics component is the active choice.

<double precision, dimension(:), allocatable => classDefault> majorMergerTime() Returns the default value for the majorMergerTime property for the mergingStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> majorMergerTimeAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the mergingStatistics class that have the desired attributes for the majorMergerTime property

<logical> majorMergerTimeIsGettable() Returns true if the majorMergerTime property is gettable for the mergingStatistics component class.

<logical> majorMergerTimeIsSettable() Specify whether the majorMergerTime property of the mergingStatistics component is settable.

<double precision, dimension(:), allocatable => classDefault> majorMergerTimeRateGet() Returns a zero rate for the majorMergerTime property for the mergingStatistics component class.

<void> majorMergerTimeSet(<double precision[:]> value) Set the majorMergerTime property of the mergingStatistics component.

<double precision> massWhenFirstIsolated() Returns the default value for the massWhenFirstIsolated property for the mergingStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> massWhenFirstIsolatedAttributeMatch(<logi [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the mergingStatistics class that have the desired attributes for the massWhenFirstIsolated property

<logical> massWhenFirstIsolatedIsGettable() Returns true if the massWhenFirstIsolated property is gettable for the mergingStatistics component class.

<logical> massWhenFirstIsolatedIsSettable() Specify whether the massWhenFirstIsolated property of the mergingStatistics component is settable.

<double precision> massWhenFirstIsolatedRateGet() Returns a zero rate for the massWhenFirstIsolated property for the mergingStatistics component class.

<void> massWhenFirstIsolatedSet(<double precision> value) Set the massWhenFirstIsolated property of the mergingStatistics component.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a null implementation of the mergingStatistics component class.

<double precision> nodeFormationTime() Returns the default value for the nodeFormationTime property for the mergingStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> nodeFormationTimeAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the mergingStatistics class that have the desired attributes for the nodeFormationTime property

<logical> nodeFormationTimeIsGettable() Returns true if the nodeFormationTime property is gettable for the mergingStatistics component class.

<logical> nodeFormationTimeIsSettable() Specify whether the nodeFormationTime property of the mergingStatistics component is settable.

<double precision> nodeFormationTimeRateGet() Returns a zero rate for the nodeFormationTime property for the mergingStatistics component class.

<void> nodeFormationTimeSet(<double precision> value) Set the nodeFormationTime property of the mergingStatistics component.

<integer> nodeHierarchyLevel() Returns the default value for the nodeHierarchyLevel property for the mergingStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> nodeHierarchyLevelAttributeMatch(<logical [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the mergingStatistics class that have the desired attributes for the nodeHierarchyLevel property

<logical> nodeHierarchyLevelIsGettable() Returns true if the nodeHierarchyLevel property is gettable for the mergingStatistics component class.

<logical> nodeHierarchyLevelIsSettable() Specify whether the nodeHierarchyLevel property of the mergingStatistics component is settable.

<integer> nodeHierarchyLevelMaximum() Returns the default value for the nodeHierarchyLevelMaximum property for the mergingStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> nodeHierarchyLevelMaximumAttributeMatch(< [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the mergingStatistics class that have the desired attributes for the nodeHierarchyLevelMaximum property

<logical> nodeHierarchyLevelMaximumIsGettable() Returns true if the nodeHierarchyLevelMaximum property is gettable for the mergingStatistics component class.

<logical> nodeHierarchyLevelMaximumIsSettable() Specify whether the nodeHierarchyLevelMaximum property of the mergingStatistics component is settable.

<integer> nodeHierarchyLevelMaximumRateGet() Returns a zero rate for the nodeHierarchyLevelMaximum property for the mergingStatistics component class.

<void> nodeHierarchyLevelMaximumSet(<integer> value) Set the nodeHierarchyLevelMaximum property of the mergingStatistics component.

<integer> nodeHierarchyLevelRateGet() Returns a zero rate for the nodeHierarchyLevel property for the mergingStatistics component class.

<void> nodeHierarchyLevelSet(<integer> value) Set the nodeHierarchyLevel property of the mergingStatistics component.

<double precision> nodeMajorMergerTime() Returns the default value for the nodeMajorMergerTime property for the mergingStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> nodeMajorMergerTimeAttributeMatch(<logica
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the mergingStatistics class that have the desired attributes for the nodeMajorMergerTime property

<logical> nodeMajorMergerTimeIsGettable() Returns true if the nodeMajorMergerTime property is gettable for the mergingStatistics component class.

<logical> nodeMajorMergerTimeIsSettable() Specify whether the nodeMajorMergerTime property of the mergingStatistics component is settable.

<double precision> nodeMajorMergerTimeRateGet() Returns a zero rate for the nodeMajorMergerTime property for the mergingStatistics component class.

<void> nodeMajorMergerTimeSet(<double precision> value) Set the nodeMajorMergerTime property of the mergingStatistics component.

<logical> nullIsActive() Return true if the null implementation of the mergingStatistics component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a mergingStatistics component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a null implementation of the mergingStatistics component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a null implementation of the mergingStatistics component.

<void> postOutput(<double precision> time→) Perform post-output processing of a mergingStatistics component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→)
Compute the gravitational potential.

<logical> recentIsActive() Return true if the recent implementation of the mergingStatistics component is the active choice.

<integer, dimension(:), allocatable => classDefault> recentMajorMergerCount() Returns the default value for the recentMajorMergerCount property for the mergingStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> recentMajorMergerCountAttributeMatch(<log [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the mergingStatistics class that have the desired attributes for the recentMajorMergerCount property

<logical> recentMajorMergerCountIsGettable() Returns true if the recentMajorMergerCount property is gettable for the mergingStatistics component class.

<logical> recentMajorMergerCountIsSettable() Specify whether the recentMajorMergerCount property of the mergingStatistics component is settable.

<integer, dimension(:), allocatable => classDefault> recentMajorMergerCountRateGet() Returns a zero rate for the recentMajorMergerCount property for the mergingStatistics component class.

<void> recentMajorMergerCountSet(<integer[:]> value) Set the recentMajorMergerCount property of the mergingStatistics component.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a null implementation of the mergingStatistics component.

<void> serializeASCII() Serialize the contents of a null implementation of the mergingStatistics component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a null implementation of the mergingStatistics component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a null implementation of the mergingStatistics component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a null implementation of the mergingStatistics component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a null implementation of the mergingStatistics component to XML.

<integer(c_size_t)> sizeOf() Return the size in bytes of a null implementation of the mergingStatistics component.

<logical> standardIsActive() Return true if the standard implementation of the mergingStatistics component is the active choice.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<type(varying_string)> type() Returns the type name for the null implementation of the mergingStatistics component class.

nodeComponentMergingStatisticsRecent

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a recent implementation of the mergingStatistics component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a recent implementation of the mergingStatistics component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a recent implementation of the mergingStatistics component from array.

<void> destroy() Finalize a recent implementation of the mergingStatistics component.

<void> dumpASCII() Dump the content of a mergingStatistics component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<double precision> galaxyMajorMergerTime() Returns the default value for the galaxyMajorMergerTime property for the mergingStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> galaxyMajorMergerTimeAttributeMatch(<logi [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the mergingStatistics class that have the desired attributes for the galaxyMajorMergerTime property

<logical> galaxyMajorMergerTimeIsGettable() Returns true if the galaxyMajorMergerTime property is gettable for the mergingStatistics component class.

<logical> galaxyMajorMergerTimeIsSettable() Specify whether the galaxyMajorMergerTime property of the mergingStatistics component is settable.

<double precision> galaxyMajorMergerTimeRateGet() Returns a zero rate for the galaxyMajorMergerTime property for the mergingStatistics component class.

<void> galaxyMajorMergerTimeSet(<double precision> value) Set the galaxyMajorMergerTime property of the mergingStatistics component.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter
[interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a recent member of the mergingStatistics component.

<logical> majorIsActive() Return true if the major implementation of the mergingStatistics component is the active choice.

<double precision, dimension(:), allocatable => classDefault> majorMergerTime() Returns the default value for the majorMergerTime property for the mergingStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> majorMergerTimeAttributeMatch(<logical>
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the mergingStatistics class that have the desired attributes for the majorMergerTime property

<logical> majorMergerTimeIsGettable() Returns true if the majorMergerTime property is gettable for the mergingStatistics component class.

<logical> majorMergerTimeIsSettable() Specify whether the majorMergerTime property of the mergingStatistics component is settable.

<double precision, dimension(:), allocatable => classDefault> majorMergerTimeRateGet()
Returns a zero rate for the majorMergerTime property for the mergingStatistics component class.

<void> majorMergerTimeSet(<double precision[:]> value) Set the majorMergerTime property of the mergingStatistics component.

<double precision> massWhenFirstIsolated() Returns the default value for the massWhenFirstIsolated property for the mergingStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> massWhenFirstIsolatedAttributeMatch(<logi
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the mergingStatistics class that have the desired attributes for the massWhenFirstIsolated property

<logical> massWhenFirstIsolatedIsGettable() Returns true if the massWhenFirstIsolated property is gettable for the mergingStatistics component class.

<logical> massWhenFirstIsolatedIsSettable() Specify whether the massWhenFirstIsolated property of the mergingStatistics component is settable.

<double precision> massWhenFirstIsolatedRateGet() Returns a zero rate for the massWhenFirstIsolated property for the mergingStatistics component class.

<void> massWhenFirstIsolatedSet(<double precision> value) Set the massWhenFirstIsolated property of the mergingStatistics component.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a recent implementation of the mergingStatistics component class.

<double precision> nodeFormationTime() Returns the default value for the nodeFormationTime property for the mergingStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> nodeFormationTimeAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the mergingStatistics class that have the desired attributes for the nodeFormationTime property

<logical> nodeFormationTimeIsGettable() Returns true if the nodeFormationTime property is gettable for the mergingStatistics component class.

<logical> nodeFormationTimeIsSettable() Specify whether the nodeFormationTime property of the mergingStatistics component is settable.

<double precision> nodeFormationTimeRateGet() Returns a zero rate for the nodeFormationTime property for the mergingStatistics component class.

<void> nodeFormationTimeSet(<double precision> value) Set the nodeFormationTime property of the mergingStatistics component.

<integer> nodeHierarchyLevel() Returns the default value for the nodeHierarchyLevel property for the mergingStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> nodeHierarchyLevelAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the mergingStatistics class that have the desired attributes for the nodeHierarchyLevel property

<logical> nodeHierarchyLevelIsGettable() Returns true if the nodeHierarchyLevel property is gettable for the mergingStatistics component class.

<logical> nodeHierarchyLevelIsSettable() Specify whether the nodeHierarchyLevel property of the mergingStatistics component is settable.

<integer> nodeHierarchyLevelMaximum() Returns the default value for the nodeHierarchyLevelMaximum property for the mergingStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> nodeHierarchyLevelMaximumAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the mergingStatistics class that have the desired attributes for the nodeHierarchyLevelMaximum property

<logical> nodeHierarchyLevelMaximumIsGettable() Returns true if the nodeHierarchyLevelMaximum property is gettable for the mergingStatistics component class.

<logical> nodeHierarchyLevelMaximumIsSettable() Specify whether the nodeHierarchyLevelMaximum property of the mergingStatistics component is settable.

<integer> nodeHierarchyLevelMaximumRateGet() Returns a zero rate for the nodeHierarchyLevelMaximum property for the mergingStatistics component class.

<void> nodeHierarchyLevelMaximumSet(<integer> value) Set the nodeHierarchyLevelMaximum property of the mergingStatistics component.

<integer> nodeHierarchyLevelRateGet() Returns a zero rate for the nodeHierarchyLevel property for the mergingStatistics component class.

<void> nodeHierarchyLevelSet(<integer> value) Set the nodeHierarchyLevel property of the mergingStatistics component.

<double precision> nodeMajorMergerTime() Returns the default value for the nodeMajorMergerTime property for the mergingStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> nodeMajorMergerTimeAttributeMatch(<logica [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the mergingStatistics class that have the desired attributes for the nodeMajorMergerTime property

<logical> nodeMajorMergerTimeIsGettable() Returns true if the nodeMajorMergerTime property is gettable for the mergingStatistics component class.

<logical> nodeMajorMergerTimeIsSettable() Specify whether the nodeMajorMergerTime property of the mergingStatistics component is settable.

<double precision> nodeMajorMergerTimeRateGet() Returns a zero rate for the nodeMajorMergerTime property for the mergingStatistics component class.

<void> nodeMajorMergerTimeSet(<double precision> value) Set the nodeMajorMergerTime property of the mergingStatistics component.

<logical> nullIsActive() Return true if the null implementation of the mergingStatistics component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_-int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a mergingStatistics component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a recent implementation of the mergingStatistics component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a recent implementation of the mergingStatistics component.

<void> postOutput(<double precision> time→) Perform post-output processing of a mergingStatistics component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<logical> recentIsActive() Return true if the recent implementation of the mergingStatistics component is the active choice.

<integer, dimension(:), allocatable => propertyValue> recentMajorMergerCount() Get the recentMajorMergerCount property of an recent implementation of the mergingStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> recentMajorMergerCountAttributeMatch(<log
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the mergingStatistics class that have the desired attributes for the recentMajorMergerCount property

<logical> recentMajorMergerCountIsGettable() Returns true if the recentMajorMergerCount property is gettable for the mergingStatistics component class.

<logical> recentMajorMergerCountIsSettable() Specify whether the recentMajorMergerCount property of the mergingStatistics component is settable.

<integer, dimension(:), allocatable => classDefault> recentMajorMergerCountRateGet() Returns a zero rate for the recentMajorMergerCount property for the mergingStatistics component class.

<void> recentMajorMergerCountSet(<integer[:]> setValue→) Set the recentMajorMergerCount property of an recent implementation of the mergingStatistics component class.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a recent implementation of the mergingStatistics component.

<void> serializeASCII() Serialize the contents of a recent implementation of the mergingStatistics component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a recent implementation of the mergingStatistics component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a recent implementation of the mergingStatistics component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a recent implementation of the mergingStatistics component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a recent implementation of the mergingStatistics component to XML.

<integer(c_size_t)> sizeOf() Return the size in bytes of a recent implementation of the mergingStatistics component.

<logical> standardIsActive() Return true if the standard implementation of the mergingStatistics component is the active choice.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<type(varying_string)> type() Returns the type name for the recent implementation of the mergingStatistics component class.

nodeComponentMergingStatisticsStandard

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a standard implementation of the mergingStatistics component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a standard implementation of the mergingStatistics component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a standard implementation of the mergingStatistics component from array.

<void> destroy() Finalize a standard implementation of the mergingStatistics component.

<void> dumpASCII() Dump the content of a mergingStatistics component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<double precision> galaxyMajorMergerTime() Get the galaxyMajorMergerTime property of an standard implementation of the mergingStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> galaxyMajorMergerTimeAttributeMatch(<logi [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the mergingStatistics class that have the desired attributes for the galaxyMajorMergerTime property

<logical> galaxyMajorMergerTimeIsGettable() Returns true if the galaxyMajorMergerTime property is gettable for the mergingStatistics component class.

<logical> galaxyMajorMergerTimeIsSettable() Specify whether the galaxyMajorMergerTime property of the mergingStatistics component is settable.

<double precision> galaxyMajorMergerTimeRateGet() Returns a zero rate for the galaxyMajorMergerTime property for the mergingStatistics component class.

<void> galaxyMajorMergerTimeSet(<double precision> setValue→) Set the galaxyMajorMergerTime property of an standard implementation of the mergingStatistics component class.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a standard member of the mergingStatistics component.

<logical> majorIsActive() Return true if the major implementation of the mergingStatistics component is the active choice.

<double precision, dimension(:), allocatable => classDefault> majorMergerTime() Returns the default value for the majorMergerTime property for the mergingStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> majorMergerTimeAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the mergingStatistics class that have the desired attributes for the majorMergerTime property

<logical> majorMergerTimeIsGettable() Returns true if the majorMergerTime property is gettable for the mergingStatistics component class.

<logical> majorMergerTimeIsSettable() Specify whether the majorMergerTime property of the mergingStatistics component is settable.

<double precision, dimension(:), allocatable => classDefault> majorMergerTimeRateGet() Returns a zero rate for the majorMergerTime property for the mergingStatistics component class.

<void> majorMergerTimeSet(<double precision[:]> value) Set the majorMergerTime property of the mergingStatistics component.

<double precision> massWhenFirstIsolated() Get the massWhenFirstIsolated property of an standard implementation of the mergingStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> massWhenFirstIsolatedAttributeMatch(<logi [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the mergingStatistics class that have the desired attributes for the massWhenFirstIsolated property

<logical> massWhenFirstIsolatedIsGettable() Returns true if the massWhenFirstIsolated property is gettable for the mergingStatistics component class.

<logical> massWhenFirstIsolatedIsSettable() Specify whether the massWhenFirstIsolated property of the mergingStatistics component is settable.

<double precision> massWhenFirstIsolatedRateGet() Returns a zero rate for the massWhenFirstIsolated property for the mergingStatistics component class.

<void> massWhenFirstIsolatedSet(<double precision> setValue→) Set the massWhenFirstIsolated property of an standard implementation of the mergingStatistics component class.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a standard implementation of the mergingStatistics component class.

<double precision> nodeFormationTime() Get the nodeFormationTime property of an standard implementation of the mergingStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> nodeFormationTimeAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the mergingStatistics class that have the desired attributes for the nodeFormationTime property

<logical> nodeFormationTimeIsGettable() Returns true if the nodeFormationTime property is gettable for the mergingStatistics component class.

<logical> nodeFormationTimeIsSettable() Specify whether the nodeFormationTime property of the mergingStatistics component is settable.

<double precision> nodeFormationTimeRateGet() Returns a zero rate for the nodeFormationTime property for the mergingStatistics component class.

<void> nodeFormationTimeSet(<double precision> setValue→) Set the nodeFormationTime property of an standard implementation of the mergingStatistics component class.

<integer> nodeHierarchyLevel() Get the value of the nodeHierarchyLevel property of the standard implementation of the mergingStatistics component using a deferred function.

<type(varying_string), allocatable, dimension(:) => matches> nodeHierarchyLevelAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the mergingStatistics class that have the desired attributes for the nodeHierarchyLevel property

<void> nodeHierarchyLevelFunction() Set the function to be used for the get method of the nodeHierarchyLevel property of the MergingStatisticsStandard component.

<logical> nodeHierarchyLevelIsAttached() Return true if the deferred function used to get the nodeHierarchyLevel property of the MergingStatisticsStandard component class has been attached.

<logical> nodeHierarchyLevelIsGettable() Returns true if the nodeHierarchyLevel property is gettable for the mergingStatistics component class.

<logical> nodeHierarchyLevelIsSettable() Specify whether the nodeHierarchyLevel property of the mergingStatistics component is settable.

<integer> nodeHierarchyLevelMaximum() Get the value of the nodeHierarchyLevelMaximum property of the standard implementation of the mergingStatistics component using a deferred function.

<type(varying_string), allocatable, dimension(:) => matches> nodeHierarchyLevelMaximumAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the mergingStatistics class that have the desired attributes for the nodeHierarchyLevelMaximum property

<void> nodeHierarchyLevelMaximumFunction() Set the function to be used for the get method of the nodeHierarchyLevelMaximum property of the MergingStatisticsStandard component.

<logical> nodeHierarchyLevelMaximumIsAttached() Return true if the deferred function used to get the nodeHierarchyLevelMaximum property of the MergingStatisticsStandard component class has been attached.

<logical> nodeHierarchyLevelMaximumIsGettable() Returns true if the nodeHierarchyLevelMaximum property is gettable for the mergingStatistics component class.

<logical> nodeHierarchyLevelMaximumIsSettable() Specify whether the nodeHierarchyLevelMaximum property of the mergingStatistics component is settable.

<integer> nodeHierarchyLevelMaximumRateGet() Returns a zero rate for the nodeHierarchyLevelMaximum property for the mergingStatistics component class.

<void> nodeHierarchyLevelMaximumSet(<integer> setValue→) Set the nodeHierarchyLevelMaximum property of an standard implementation of the mergingStatistics component class.

<integer> nodeHierarchyLevelMaximumValue() Get the nodeHierarchyLevelMaximum property of an standard implementation of the mergingStatistics component class.

<integer> nodeHierarchyLevelRateGet() Returns a zero rate for the nodeHierarchyLevel property for the mergingStatistics component class.

<void> nodeHierarchyLevelSet(<integer> setValue→) Set the nodeHierarchyLevel property of an standard implementation of the mergingStatistics component class.

<integer> nodeHierarchyLevelValue() Get the nodeHierarchyLevel property of an standard implementation of the mergingStatistics component class.

<double precision> nodeMajorMergerTime() Get the nodeMajorMergerTime property of an standard implementation of the mergingStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> nodeMajorMergerTimeAttributeMatch(<logica [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the mergingStatistics class that have the desired attributes for the nodeMajorMergerTime property

<logical> nodeMajorMergerTimeIsGettable() Returns true if the nodeMajorMergerTime property is gettable for the mergingStatistics component class.

<logical> nodeMajorMergerTimeIsSettable() Specify whether the nodeMajorMergerTime property of the mergingStatistics component is settable.

<double precision> nodeMajorMergerTimeRateGet() Returns a zero rate for the nodeMajorMergerTime property for the mergingStatistics component class.

<void> nodeMajorMergerTimeSet(<double precision> setValue→) Set the nodeMajorMergerTime property of an standard implementation of the mergingStatistics component class.

<logical> nullIsActive() Return true if the null implementation of the mergingStatistics component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a mergingStatistics component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a standard implementation of the mergingStatistics component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a standard implementation of the mergingStatistics component.

<void> postOutput(<double precision> time→) Perform post-output processing of a mergingStatistics component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<logical> recentIsActive() Return true if the recent implementation of the mergingStatistics component is the active choice.

<integer, dimension(:), allocatable => classDefault> recentMajorMergerCount() Returns the default value for the recentMajorMergerCount property for the mergingStatistics component class.

<type(varying_string), allocatable, dimension(:) => matches> recentMajorMergerCountAttributeMatch(<log [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the mergingStatistics class that have the desired attributes for the recentMajorMergerCount property

<logical> recentMajorMergerCountIsGettable() Returns true if the recentMajorMergerCount property is gettable for the mergingStatistics component class.

<logical> recentMajorMergerCountIsSettable() Specify whether the recentMajorMergerCount property of the mergingStatistics component is settable.

<integer, dimension(:), allocatable => classDefault> recentMajorMergerCountRateGet() Returns a zero rate for the recentMajorMergerCount property for the mergingStatistics component class.

<void> recentMajorMergerCountSet(<integer[:]> value) Set the recentMajorMergerCount property of the mergingStatistics component.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a standard implementation of the mergingStatistics component.

<void> serializeASCII() Serialize the contents of a standard implementation of the mergingStatistics component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a standard implementation of the mergingStatistics component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a standard implementation of the mergingStatistics component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a standard implementation of the mergingStatistics component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a standard implementation of the mergingStatistics component to XML.

<integer(c_size_t)> sizeOf() Return the size in bytes of a standard implementation of the mergingStatistics component.

<logical> standardIsActive() Return true if the standard implementation of the mergingStatistics component is the active choice.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<type(varying_string)> type() Returns the type name for the standard implementation of the mergingStatistics component class.

nodeComponentNBody

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a generic nBody component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Read properties from raw file.

<void> deserializeValues(<double(:)> array→, <integer> propertyType→) Deserialize the evolvable quantities from an array.

<void> destroy() Finalize a generic nBody component.

<void> dumpASCII() Dump the content of a nBody component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<logical> genericIsActive() Return true if the generic implementation of the nBody component is the active choice.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a generic nBody component.

<integer(kind=kind_int8), dimension(:), allocatable => classDefault> integers() Returns the default value for the integers property for the nBody component class.

<type(varying_string), allocatable, dimension(:) => matches> integersAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the nBody class that have the desired attributes for the integers property

<logical> integersIsGettable() Returns true if the integers property is gettable for the nBody component class.

<logical> integersIsSettable() Specify whether the integers property of the nBody component is settable.

<integer(kind=kind_int8), dimension(:), allocatable => classDefault> integersRateGet() Returns a zero rate for the integers property for the nBody component class.

<void> integersSet(<integer(kind=kind_int8)[:]> value) Set the integers property of the nBody component.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a nodeComponent object.

<logical> nullIsActive() Return true if the null implementation of the nBody component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a nBody component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a generic nBody component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Establish the names of properties to output for a generic nBody component.

<void> postOutput(<double precision> time→) Perform post-output processing of a nBody component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<double precision, dimension(:), allocatable => classDefault> reals() Returns the default value for the reals property for the nBody component class.

<type(varying_string), allocatable, dimension(:) => matches> realsAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the nBody class that have the desired attributes for the reals property

<logical> realsIsGettable() Returns true if the reals property is gettable for the nBody component class.

<logical> realsIsSettable() Specify whether the reals property of the nBody component is settable.

<double precision, dimension(:), allocatable => classDefault> realsRateGet() Returns a zero rate for the reals property for the nBody component class.

<void> realsSet(<double precision[:]> value) Set the reals property of the nBody component.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets() Set offsets into serialization arrays.

<void> serializeASCII() Serialize the content of a nBody component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the number of evolvable quantities to be evolved.

<void> serializeRaw(<integer> fileHandle→) Generate a binary dump of all properties.

<void> serializeValues(<double(:)> array←, <integer> propertyType→) Serialize the evolvable quantities to an array.

<void> serializeXML() Generate an XML dump of all properties.

<integer(c_size_t)> sizeOf() Return the size in bytes of a nodeComponentNBody component.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<type(varying_string)> type() Returns the type name for the nBody component class.

nodeComponentNBodyGeneric

<integer> addIntegerProperty(<character(len=*)> propertyname→) Call the deferred function for the addIntegerProperty method of the nBody component class if it has been set.

<void> addIntegerPropertyFunction() Set the function to be used for the addIntegerProperty method of the generic implementation of the nBody component class.

<logical> addIntegerPropertyFunctionIsSet() Return true if the deferred function for the addIntegerProperty method of the generic implementation of the nBody component class has been set.

<integer> addRealProperty(<character(len=*)> propertyname→) Call the deferred function for the addRealProperty method of the nBody component class if it has been set.

<void> addRealPropertyFunction() Set the function to be used for the addRealProperty method of the generic implementation of the nBody component class.

<logical> addRealPropertyFunctionIsSet() Return true if the deferred function for the addRealProperty method of the generic implementation of the nBody component class has been set.

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a generic implementation of the nBody component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a generic implementation of the nBody component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a generic implementation of the nBody component from array.

<void> destroy() Finalize a generic implementation of the nBody component.

<void> dumpASCII() Dump the content of a nBody component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<logical> genericIsActive() Return true if the generic implementation of the nBody component is the active choice.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter
[interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a generic member of the nBody component.

<integer(kind=kind_int8), dimension(:), allocatable => propertyValue> integers() Get the integers property of an generic implementation of the nBody component class.

<type(varying_string), allocatable, dimension(:) => matches> integersAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the nBody class that have the desired attributes for the integers property

<logical> integersIsGettable() Returns true if the integers property is gettable for the nBody component class.

<logical> integersIsSettable() Specify whether the integers property of the nBody component is settable.

<integer(kind=kind_int8), dimension(:), allocatable => classDefault> integersRateGet() Returns a zero rate for the integers property for the nBody component class.

<void> integersSet(<integer(kind=kind_int8)[:]> setValue→) Set the integers property of an generic implementation of the nBody component class.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a generic implementation of the nBody component class.

<logical> nullIsActive() Return true if the null implementation of the nBody component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a nBody component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a generic implementation of the nBody component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a generic implementation of the nBody component.

<void> postOutput(<double precision> time→) Perform post-output processing of a nBody component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<double precision, dimension(:), allocatable => propertyValue> reals() Get the reals property of an generic implementation of the nBody component class.

<type(varying_string), allocatable, dimension(:) => matches> realsAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the nBody class that have the desired attributes for the reals property

<logical> realsIsGettable() Returns true if the reals property is gettable for the nBody component class.

<logical> realsIsSettable() Specify whether the reals property of the nBody component is settable.

<double precision, dimension(:), allocatable => classDefault> realsRateGet() Returns a zero rate for the reals property for the nBody component class.

<void> realsSet(<double precision[:]> setValue→) Set the reals property of an generic implementation of the nBody component class.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a generic implementation of the nBody component.

<void> serializeASCII() Serialize the contents of a generic implementation of the nBody component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a generic implementation of the nBody component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a generic implementation of the nBody component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a generic implementation of the nBody component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a generic implementation of the nBody component to XML.

<void> setIntegerProperty(<integer> propertyindex→, <integer(kind_int8)> propertyvalue→) Call the deferred function for the setIntegerProperty method of the nBody component class if it has been set.

<void> setIntegerPropertyFunction() Set the function to be used for the setIntegerProperty method of the generic implementation of the nBody component class.

<logical> setIntegerPropertyFunctionIsSet() Return true if the deferred function for the setIntegerProperty method of the generic implementation of the nBody component class has been set.

<void> setRealProperty(<integer> propertyindex→, <double precision> propertyvalue→) Call the deferred function for the setRealProperty method of the nBody component class if it has been set.

<void> setRealPropertyFunction() Set the function to be used for the setRealProperty method of the generic implementation of the nBody component class.

<logical> setRealPropertyFunctionIsSet() Return true if the deferred function for the setRealProperty method of the generic implementation of the nBody component class has been set.

<integer(c_size_t)> sizeOf() Return the size in bytes of a generic implementation of the nBody component.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<type(varying_string)> type() Returns the type name for the generic implementation of the nBody component class.

nodeComponentNBodyNull

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a null implementation of the nBody component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a null implementation of the nBody component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a null implementation of the nBody component from array.

<void> destroy() Finalize a null implementation of the nBody component.

<void> dumpASCII() Dump the content of a nBody component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<logical> genericIsActive() Return true if the generic implementation of the nBody component is the active choice.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a null member of the nBody component.

<integer(kind=kind_int8), dimension(:), allocatable => classDefault> integers() Returns the default value for the integers property for the nBody component class.

<type(varying_string), allocatable, dimension(:) => matches> integersAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the nBody class that have the desired attributes for the integers property

<logical> integersIsGettable() Returns true if the integers property is gettable for the nBody component class.

<logical> integersIsSettable() Specify whether the integers property of the nBody component is settable.

<integer(kind=kind_int8), dimension(:), allocatable => classDefault> integersRateGet() Returns a zero rate for the integers property for the nBody component class.

<void> integersSet(<integer(kind=kind_int8)[:]> value) Set the integers property of the nBody component.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a null implementation of the nBody component class.

<logical> nullIsActive() Return true if the null implementation of the nBody component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a nBody component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a null implementation of the nBody component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision>

time→, <integer> instance→) Return the names of properties to output for a null implementation of the nBody component.

<void> postOutput(<double precision> time→) Perform post-output processing of a nBody component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<double precision, dimension(:), allocatable => classDefault> reals() Returns the default value for the reals property for the nBody component class.

<type(varying_string), allocatable, dimension(:) => matches> realsAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the nBody class that have the desired attributes for the reals property

<logical> realsIsGettable() Returns true if the reals property is gettable for the nBody component class.

<logical> realsIsSettable() Specify whether the reals property of the nBody component is settable.

<double precision, dimension(:), allocatable => classDefault> realsRateGet() Returns a zero rate for the reals property for the nBody component class.

<void> realsSet(<double precision[:]> value) Set the reals property of the nBody component.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a null implementation of the nBody component.

<void> serializeASCII() Serialize the contents of a null implementation of the nBody component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a null implementation of the nBody component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a null implementation of the nBody component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a null implementation of the nBody component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a null implementation of the nBody component to XML.

<integer(c_size_t)> sizeOf() Return the size in bytes of a null implementation of the nBody component.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<type(varying_string)> type() Returns the type name for the null implementation of the nBody component class.

nodeComponentPosition

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a generic position component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Read properties from raw file.

<void> deserializeValues(<double(:)> array→, <integer> propertyType→) Deserialize the evolvable quantities from an array.

<void> destroy() Finalize a generic position component.

<void> dumpASCII() Dump the content of a position component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a generic position component.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a nodeComponent object.

<logical> nullIsActive() Return true if the null implementation of the position component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a position component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a generic position component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Establish the names of properties to output for a generic position component.

<double precision, dimension(:), allocatable => classDefault> position() Returns the default value for the position property for the position component class.

<type(varying_string), allocatable, dimension(:) => matches> positionAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the position class that have the desired attributes for the position property

<type(history)> positionHistory() Returns the default value for the positionHistory property for the position component class.

<type(varying_string), allocatable, dimension(:) => matches> positionHistoryAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the position class that have the desired attributes for the positionHistory property

<logical> positionHistoryIsGettable() Returns true if the positionHistory property is gettable for the position component class.

<logical> positionHistoryIsSettable() Specify whether the positionHistory property of the position component is settable.

<type(history)> positionHistoryRateGet() Returns a zero rate for the positionHistory property for the position component class.

<void> positionHistorySet(<type(history)> value) Set the positionHistory property of the position component.

<logical> positionIsGettable() Returns true if the position property is gettable for the position component class.

<logical> positionIsSettable() Specify whether the position property of the position component is settable.

<double precision, dimension(:), allocatable => classDefault> positionOrphan() Returns the default value for the positionOrphan property for the position component class.

<type(varying_string), allocatable, dimension(:) => matches> positionOrphanAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the position class that have the desired attributes for the positionOrphan property

<logical> positionOrphanIsGettable() Returns true if the positionOrphan property is gettable for the position component class.

<logical> positionOrphanIsSettable() Specify whether the positionOrphan property of the position component is settable.

<double precision, dimension(:), allocatable => classDefault> positionOrphanRateGet() Returns a zero rate for the positionOrphan property for the position component class.

<void> positionOrphanSet(<double precision[:]> value) Set the positionOrphan property of the position component.

<double precision, dimension(:), allocatable => classDefault> positionRateGet() Returns a zero rate for the position property for the position component class.

<void> positionSet(<double precision[:]> value) Set the position property of the position component.

<void> postOutput(<double precision> time→) Perform post-output processing of a position component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<logical> presetIsActive() Return true if the preset implementation of the position component is the active choice.

<logical> presetOrphansIsActive() Return true if the presetOrphans implementation of the position component is the active choice.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets() Set offsets into serialization arrays.

<void> serializeASCII() Serialize the content of a position component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the number of evolvable quantities to be evolved.

<void> serializeRaw(<integer> fileHandle→) Generate a binary dump of all properties.

<void> serializeValues(<double(:)> array←, <integer> propertyType→) Serialize the evolvable quantities to an array.

<void> serializeXML() Generate an XML dump of all properties.

<integer(c_size_t)> sizeOf() Return the size in bytes of a nodeComponentPosition component.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<double precision> timeAssign() Returns the default value for the timeAssign property for the position component class.

<type(varying_string), allocatable, dimension(:) => matches> timeAssignAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the position class that have the desired attributes for the timeAssign property

<logical> timeAssignIsGettable() Returns true if the timeAssign property is gettable for the position component class.

<logical> timeAssignIsSettable() Specify whether the timeAssign property of the position component is settable.

<double precision> timeAssignRateGet() Returns a zero rate for the timeAssign property for the position component class.

<void> timeAssignSet(<double precision> value) Set the timeAssign property of the position component.

<logical> traceDarkMatterIsActive() Return true if the traceDarkMatter implementation of the position component is the active choice.

<type(varying_string)> type() Returns the type name for the position component class.

<double precision, dimension(:), allocatable => classDefault> velocity() Returns the default value for the velocity property for the position component class.

<type(varying_string), allocatable, dimension(:) => matches> velocityAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the position class that have the desired attributes for the velocity property

<logical> velocityIsGettable() Returns true if the velocity property is gettable for the position component class.

<logical> velocityIsSettable() Specify whether the velocity property of the position component is settable.

<double precision, dimension(:), allocatable => classDefault> velocityOrphan() Returns the default value for the velocityOrphan property for the position component class.

<type(varying_string), allocatable, dimension(:) => matches> velocityOrphanAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the position class that have the desired attributes for the velocityOrphan property

<logical> velocityOrphanIsGettable() Returns true if the velocityOrphan property is gettable for the position component class.

<logical> velocityOrphanIsSettable() Specify whether the velocityOrphan property of the position component is settable.

<double precision, dimension(:), allocatable => classDefault> velocityOrphanRateGet() Returns a zero rate for the velocityOrphan property for the position component class.

<void> velocityOrphanSet(<double precision[:]> value) Set the velocityOrphan property of the position component.

<double precision, dimension(:), allocatable => classDefault> velocityRateGet() Returns a zero rate for the velocity property for the position component class.

<void> velocitySet(<double precision[:]> value) Set the velocity property of the position component.

nodeComponentPositionNull

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a null implementation of the position component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a null implementation of the position component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a null implementation of the position component from array.

<void> destroy() Finalize a null implementation of the position component.

<void> dumpASCII() Dump the content of a position component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a null member of the position component.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a null implementation of the position component class.

<logical> nullIsActive() Return true if the null implementation of the position component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a position component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a null implementation of the position component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a null implementation of the position component.

<double precision, dimension(:), allocatable => classDefault> position() Returns the default value for the position property for the position component class.

<type(varying_string), allocatable, dimension(:) => matches> positionAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the position class that have the desired attributes for the position property

<type(history)> positionHistory() Returns the default value for the positionHistory property for the position component class.

<type(varying_string), allocatable, dimension(:) => matches> positionHistoryAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the position class that have the desired attributes for the positionHistory property

<logical> positionHistoryIsGettable() Returns true if the positionHistory property is gettable for the position component class.

<logical> positionHistoryIsSettable() Specify whether the positionHistory property of the position component is settable.

<type(history)> positionHistoryRateGet() Returns a zero rate for the positionHistory property for the position component class.

<void> positionHistorySet(<type(history)> value) Set the positionHistory property of the position component.

<logical> positionIsGettable() Returns true if the position property is gettable for the position component class.

<logical> positionIsSettable() Specify whether the position property of the position component is settable.

<double precision, dimension(:), allocatable => classDefault> positionOrphan() Returns the default value for the positionOrphan property for the position component class.

<type(varying_string), allocatable, dimension(:) => matches> positionOrphanAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the position class that have the desired attributes for the positionOrphan property

<logical> positionOrphanIsGettable() Returns true if the positionOrphan property is gettable for the position component class.

<logical> positionOrphanIsSettable() Specify whether the positionOrphan property of the position component is settable.

<double precision, dimension(:), allocatable => classDefault> positionOrphanRateGet() Returns a zero rate for the positionOrphan property for the position component class.

<void> positionOrphanSet(<double precision[:]> value) Set the positionOrphan property of the position component.

<double precision, dimension(:), allocatable => classDefault> positionRateGet() Returns a zero rate for the position property for the position component class.

<void> positionSet(<double precision[:]> value) Set the position property of the position component.

<void> postOutput(<double precision> time→) Perform post-output processing of a position component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<logical> presetIsActive() Return true if the preset implementation of the position component is the active choice.

<logical> presetOrphansIsActive() Return true if the presetOrphans implementation of the position component is the active choice.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a null implementation of the position component.

<void> serializeASCII() Serialize the contents of a null implementation of the position component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a null implementation of the position component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a null implementation of the position component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a null implementation of the position component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a null implementation of the position component to XML.

<integer(c_size_t)> sizeOf() Return the size in bytes of a null implementation of the position component.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<double precision> timeAssign() Returns the default value for the timeAssign property for the position component class.

<type(varying_string), allocatable, dimension(:) => matches> timeAssignAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the position class that have the desired attributes for the timeAssign property

<logical> timeAssignIsGettable() Returns true if the timeAssign property is gettable for the position component class.

<logical> timeAssignIsSettable() Specify whether the timeAssign property of the position component is settable.

<double precision> timeAssignRateGet() Returns a zero rate for the timeAssign property for the position component class.

<void> timeAssignSet(<double precision> value) Set the timeAssign property of the position component.

<logical> traceDarkMatterIsActive() Return true if the traceDarkMatter implementation of the position component is the active choice.

<type(varying_string)> type() Returns the type name for the null implementation of the position component class.

<double precision, dimension(:), allocatable => classDefault> velocity() Returns the default value for the velocity property for the position component class.

<type(varying_string), allocatable, dimension(:) => matches> velocityAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the position class that have the desired attributes for the velocity property

<logical> velocityIsGettable() Returns true if the velocity property is gettable for the position component class.

<logical> velocityIsSettable() Specify whether the velocity property of the position component is settable.

<double precision, dimension(:), allocatable => classDefault> velocityOrphan() Returns the default value for the velocityOrphan property for the position component class.

<type(varying_string), allocatable, dimension(:) => matches> velocityOrphanAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the position class that have the desired attributes for the velocityOrphan property

<logical> velocityOrphanIsGettable() Returns true if the velocityOrphan property is gettable for the position component class.

<logical> velocityOrphanIsSettable() Specify whether the velocityOrphan property of the position component is settable.

<double precision, dimension(:), allocatable => classDefault> velocityOrphanRateGet() Returns a zero rate for the velocityOrphan property for the position component class.

<void> velocityOrphanSet(<double precision[:]> value) Set the velocityOrphan property of the position component.

<double precision, dimension(:), allocatable => classDefault> velocityRateGet() Returns a zero rate for the velocity property for the position component class.

<void> velocitySet(<double precision[:]> value) Set the velocity property of the position component.

nodeComponentPositionPreset

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a preset implementation of the position component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a preset implementation of the position component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a preset implementation of the position component from array.

<void> destroy() Finalize a preset implementation of the position component.

<void> dumpASCII() Dump the content of a position component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a preset member of the position component.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a preset implementation of the position component class.

<logical> nullIsActive() Return true if the null implementation of the position component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a position component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a preset implementation of the position component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a preset implementation of the position component.

<double precision, dimension(:), allocatable => classDefault> position() Returns the default value for the position property for the position component class.

<type(varying_string), allocatable, dimension(:) => matches> positionAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the position class that have the desired attributes for the position property

<type(history)> positionHistory() Get the positionHistory property of an preset implementation of the position component class.

<type(varying_string), allocatable, dimension(:) => matches> positionHistoryAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the position class that have the desired attributes for the positionHistory property

<logical> positionHistoryIsGettable() Returns true if the positionHistory property is gettable for the position component class.

<logical> positionHistoryIsSettable() Specify whether the positionHistory property of the position component is settable.

<type(history)> positionHistoryRateGet() Returns a zero rate for the positionHistory property for the position component class.

<void> positionHistorySet(<type(history)> setValue→) Set the positionHistory property of an preset implementation of the position component class.

<logical> positionIsGettable() Returns true if the position property is gettable for the position component class.

<logical> positionIsSettable() Specify whether the position property of the position component is settable.

<double precision, dimension(:), allocatable => classDefault> positionOrphan() Returns the default value for the positionOrphan property for the position component class.

<type(varying_string), allocatable, dimension(:) => matches> positionOrphanAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the position class that have the desired attributes for the positionOrphan property

<logical> positionOrphanIsGettable() Returns true if the positionOrphan property is gettable for the position component class.

<logical> positionOrphanIsSettable() Specify whether the positionOrphan property of the position component is settable.

<double precision, dimension(:), allocatable => classDefault> positionOrphanRateGet() Returns a zero rate for the positionOrphan property for the position component class.

<void> positionOrphanSet(<double precision[:]> value) Set the positionOrphan property of the position component.

<double precision, dimension(:), allocatable => classDefault> positionRateGet() Returns a zero rate for the position property for the position component class.

<void> positionSet(<double precision[:]> setValue→) Set the position property of an preset implementation of the position component class.

<void> postOutput(<double precision> time→) Perform post-output processing of a position component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<logical> presetIsActive() Return true if the preset implementation of the position component is the active choice.

<logical> presetOrphansIsActive() Return true if the presetOrphans implementation of the position component is the active choice.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a preset implementation of the position component.

<void> serializeASCII() Serialize the contents of a preset implementation of the position component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a preset implementation of the position component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a preset implementation of the position component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a preset implementation of the position component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a preset implementation of the position component to XML.

<integer(c_size_t)> sizeOf() Return the size in bytes of a preset implementation of the position component.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<double precision> timeAssign() Returns the default value for the timeAssign property for the position component class.

<type(varying_string), allocatable, dimension(:) => matches> timeAssignAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the position class that have the desired attributes for the timeAssign property

<logical> timeAssignIsGettable() Returns true if the timeAssign property is gettable for the position component class.

<logical> timeAssignIsSettable() Specify whether the timeAssign property of the position component is settable.

<double precision> timeAssignRateGet() Returns a zero rate for the timeAssign property for the position component class.

<void> timeAssignSet(<double precision> value) Set the timeAssign property of the position component.

<logical> traceDarkMatterIsActive() Return true if the traceDarkMatter implementation of the position component is the active choice.

<type(varying_string)> type() Returns the type name for the preset implementation of the position component class.

<double precision, dimension(:), allocatable => classDefault> velocity() Returns the default value for the velocity property for the position component class.

<type(varying_string), allocatable, dimension(:) => matches> velocityAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the position class that have the desired attributes for the velocity property

<logical> velocityIsGettable() Returns true if the velocity property is gettable for the position component class.

<logical> velocityIsSettable() Specify whether the velocity property of the position component is settable.

<double precision, dimension(:), allocatable => classDefault> velocityOrphan() Returns the default value for the velocityOrphan property for the position component class.

<type(varying_string), allocatable, dimension(:) => matches> velocityOrphanAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the position class that have the desired attributes for the velocityOrphan property

<logical> velocityOrphanIsGettable() Returns true if the velocityOrphan property is gettable for the position component class.

<logical> velocityOrphanIsSettable() Specify whether the velocityOrphan property of the position component is settable.

<double precision, dimension(:), allocatable => classDefault> velocityOrphanRateGet() Returns a zero rate for the velocityOrphan property for the position component class.

<void> velocityOrphanSet(<double precision[:]> value) Set the velocityOrphan property of the position component.

<double precision, dimension(:), allocatable => classDefault> velocityRateGet() Returns a zero rate for the velocity property for the position component class.

<void> velocitySet(<double precision[:]> setValue→) Set the velocity property of an preset implementation of the position component class.

nodeComponentPositionPresetOrphans

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a presetOrphans implementation of the position component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a presetOrphans implementation of the position component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a presetOrphans implementation of the position component from array.

<void> destroy() Finalize a presetOrphans implementation of the position component.

<void> dumpASCII() Dump the content of a position component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a presetOrphans member of the position component.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a presetOrphans implementation of the position component class.

<logical> nullIsActive() Return true if the null implementation of the position component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a position component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a presetOrphans implementation of the position component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a presetOrphans implementation of the position component.

<double precision, dimension(:), allocatable => classDefault> position() Returns the default value for the position property for the position component class.

<type(varying_string), allocatable, dimension(:) => matches> positionAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the position class that have the desired attributes for the position property

<type(history)> positionHistory() Get the positionHistory property of an preset implementation of the position component class.

<type(varying_string), allocatable, dimension(:) => matches> positionHistoryAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the position class that have the desired attributes for the positionHistory property

<logical> positionHistoryIsGettable() Returns true if the positionHistory property is gettable for the position component class.

<logical> positionHistoryIsSettable() Specify whether the positionHistory property of the position component is settable.

<type(history)> positionHistoryRateGet() Returns a zero rate for the positionHistory property for the position component class.

<void> positionHistorySet(<type(history)> setValue→) Set the positionHistory property of an preset implementation of the position component class.

<logical> positionIsGettable() Returns true if the position property is gettable for the position component class.

<logical> positionIsSettable() Specify whether the position property of the position component is settable.

<double precision, dimension(:), allocatable => propertyValue> positionOrphan() Get the value of the positionOrphan property of the presetOrphans implementation of the position component using a deferred function.

<type(varying_string), allocatable, dimension(:) => matches> positionOrphanAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the position class that have the desired attributes for the positionOrphan property

<void> positionOrphanFunction() Set the function to be used for the get method of the positionOrphan property of the PositionPresetOrphans component.

<logical> positionOrphanIsAttached() Return true if the deferred function used to get the positionOrphan property of the PositionPresetOrphans component class has been attached.

<logical> positionOrphanIsGettable() Returns true if the positionOrphan property is gettable for the position component class.

<logical> positionOrphanIsSettable() Specify whether the positionOrphan property of the position component is settable.

<double precision, dimension(:), allocatable => classDefault> positionOrphanRateGet() Returns a zero rate for the positionOrphan property for the position component class.

<void> positionOrphanSet(<double precision[:]> setValue→) Set the positionOrphan property of an presetOrphans implementation of the position component class.

<double precision, dimension(:), allocatable => propertyValue> positionOrphanValue() Get the positionOrphan property of an presetOrphans implementation of the position component class.

<double precision, dimension(:), allocatable => classDefault> positionRateGet() Returns a zero rate for the position property for the position component class.

<void> positionSet(<double precision[:]> setValue→) Set the position property of an presetOrphans implementation of the position component class.

<void> postOutput(<double precision> time→) Perform post-output processing for a presetOrphans implementation of the position component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<logical> presetIsActive() Return true if the preset implementation of the position component is the active choice.

<logical> presetOrphansIsActive() Return true if the presetOrphans implementation of the position component is the active choice.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a presetOrphans implementation of the position component.

<void> serializeASCII() Serialize the contents of a presetOrphans implementation of the position component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a presetOrphans implementation of the position component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a presetOrphans implementation of the position component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a presetOrphans implementation of the position component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a presetOrphans implementation of the position component to XML.

<integer(c_size_t)> sizeOf() Return the size in bytes of a presetOrphans implementation of the position component.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<double precision> timeAssign() Get the timeAssign property of an presetOrphans implementation of the position component class.

<type(varying_string), allocatable, dimension(:) => matches> timeAssignAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the position class that have the desired attributes for the timeAssign property

<logical> timeAssignIsGettable() Returns true if the timeAssign property is gettable for the position component class.

<logical> timeAssignIsSettable() Specify whether the timeAssign property of the position component is settable.

<double precision> timeAssignRateGet() Returns a zero rate for the timeAssign property for the position component class.

<void> timeAssignSet(<double precision> setValue→) Set the timeAssign property of an presetOrphans implementation of the position component class.

<logical> traceDarkMatterIsActive() Return true if the traceDarkMatter implementation of the position component is the active choice.

<type(varying_string)> type() Returns the type name for the presetOrphans implementation of the position component class.

<double precision, dimension(:), allocatable => classDefault> velocity() Returns the default value for the velocity property for the position component class.

<type(varying_string), allocatable, dimension(:) => matches> velocityAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the position class that have the desired attributes for the velocity property

<logical> velocityIsGettable() Returns true if the velocity property is gettable for the position component class.

<logical> velocityIsSettable() Specify whether the velocity property of the position component is settable.

<double precision, dimension(:), allocatable => propertyValue> velocityOrphan() Get the value of the velocityOrphan property of the presetOrphans implementation of the position component using a deferred function.

<type(varying_string), allocatable, dimension(:) => matches> velocityOrphanAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the position class that have the desired attributes for the velocityOrphan property

<void> velocityOrphanFunction() Set the function to be used for the get method of the velocityOrphan property of the PositionPresetOrphans component.

<logical> velocityOrphanIsAttached() Return true if the deferred function used to get the velocityOrphan property of the PositionPresetOrphans component class has been attached.

<logical> velocityOrphanIsGettable() Returns true if the velocityOrphan property is gettable for the position component class.

<logical> velocityOrphanIsSettable() Specify whether the velocityOrphan property of the position component is settable.

<double precision, dimension(:), allocatable => classDefault> velocityOrphanRateGet() Returns a zero rate for the velocityOrphan property for the position component class.

<void> velocityOrphanSet(<double precision[:]> setValue→) Set the velocityOrphan property of an presetOrphans implementation of the position component class.

<double precision, dimension(:), allocatable => propertyValue> velocityOrphanValue() Get the velocityOrphan property of an presetOrphans implementation of the position component class.

<double precision, dimension(:), allocatable => classDefault> velocityRateGet() Returns a zero rate for the velocity property for the position component class.

<void> velocitySet(<double precision[:]> setValue→) Set the velocity property of an presetOrphans implementation of the position component class.

nodeComponentPositionTraceDarkMatter

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a traceDarkMatter implementation of the position component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a traceDarkMatter implementation of the position component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a traceDarkMatter implementation of the position component from array.

<void> destroy() Finalize a traceDarkMatter implementation of the position component.

<void> dumpASCII() Dump the content of a position component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a traceDarkMatter member of the position component.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a traceDarkMatter implementation of the position component class.

<logical> nullIsActive() Return true if the null implementation of the position component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a position component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a traceDarkMatter implementation of the position component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a traceDarkMatter implementation of the position component.

<double precision, dimension(:), allocatable => propertyValue> position() Get the position property of an traceDarkMatter implementation of the position component class.

<type(varying_string), allocatable, dimension(:) => matches> positionAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the position class that have the desired attributes for the position property

<type(history)> positionHistory() Returns the default value for the positionHistory property for the position component class.

<type(varying_string), allocatable, dimension(:) => matches> positionHistoryAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the position class that have the desired attributes for the positionHistory property

<logical> positionHistoryIsGettable() Returns true if the positionHistory property is gettable for the position component class.

<logical> positionHistoryIsSettable() Specify whether the positionHistory property of the position component is settable.

<type(history)> positionHistoryRateGet() Returns a zero rate for the positionHistory property for the position component class.

<void> positionHistorySet(<type(history)> value) Set the positionHistory property of the position component.

<logical> positionIsGettable() Returns true if the position property is gettable for the position component class.

<logical> positionIsSettable() Specify whether the position property of the position component is settable.

<double precision, dimension(:), allocatable => classDefault> positionOrphan() Returns the default value for the positionOrphan property for the position component class.

<type(varying_string), allocatable, dimension(:) => matches> positionOrphanAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the position class that have the desired attributes for the positionOrphan property

<logical> positionOrphanIsGettable() Returns true if the positionOrphan property is gettable for the position component class.

<logical> positionOrphanIsSettable() Specify whether the positionOrphan property of the position component is settable.

<double precision, dimension(:), allocatable => classDefault> positionOrphanRateGet() Returns a zero rate for the positionOrphan property for the position component class.

<void> positionOrphanSet(<double precision[:]> value) Set the positionOrphan property of the position component.

<double precision, dimension(:), allocatable => classDefault> positionRateGet() Returns a zero rate for the position property for the position component class.

<void> positionSet(<double precision[:]> setValue→) Set the position property of an traceDarkMatter implementation of the position component class.

<void> postOutput(<double precision> time→) Perform post-output processing of a position component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<logical> presetIsActive() Return true if the preset implementation of the position component is the active choice.

<logical> presetOrphansIsActive() Return true if the presetOrphans implementation of the position component is the active choice.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a traceDarkMatter implementation of the position component.

<void> serializeASCII() Serialize the contents of a traceDarkMatter implementation of the position component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a traceDarkMatter implementation of the position component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a traceDarkMatter implementation of the position component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a traceDarkMatter implementation of the position component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a traceDarkMatter implementation of the position component to XML.

<integer(c_size_t)> sizeOf() Return the size in bytes of a traceDarkMatter implementation of the position component.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<double precision> timeAssign() Returns the default value for the timeAssign property for the position component class.

<type(varying_string), allocatable, dimension(:) => matches> timeAssignAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the position class that have the desired attributes for the timeAssign property

<logical> timeAssignIsGettable() Returns true if the timeAssign property is gettable for the position component class.

<logical> timeAssignIsSettable() Specify whether the timeAssign property of the position component is settable.

<double precision> timeAssignRateGet() Returns a zero rate for the timeAssign property for the position component class.

<void> timeAssignSet(<double precision> value) Set the timeAssign property of the position component.

<logical> traceDarkMatterIsActive() Return true if the traceDarkMatter implementation of the position component is the active choice.

<type(varying_string)> type() Returns the type name for the traceDarkMatter implementation of the position component class.

<double precision, dimension(:), allocatable => classDefault> velocity() Returns the default value for the velocity property for the position component class.

<type(varying_string), allocatable, dimension(:) => matches> velocityAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the position class that have the desired attributes for the velocity property

<logical> velocityIsGettable() Returns true if the velocity property is gettable for the position component class.

<logical> velocityIsSettable() Specify whether the velocity property of the position component is settable.

<double precision, dimension(:), allocatable => classDefault> velocityOrphan() Returns the default value for the velocityOrphan property for the position component class.

<type(varying_string), allocatable, dimension(:) => matches> velocityOrphanAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the position class that have the desired attributes for the velocityOrphan property

<logical> velocityOrphanIsGettable() Returns true if the velocityOrphan property is gettable for the position component class.

<logical> velocityOrphanIsSettable() Specify whether the velocityOrphan property of the position component is settable.

<double precision, dimension(:), allocatable => classDefault> velocityOrphanRateGet() Returns a zero rate for the velocityOrphan property for the position component class.

<void> velocityOrphanSet(<double precision[:]> value) Set the velocityOrphan property of the position component.

<double precision, dimension(:), allocatable => classDefault> velocityRateGet() Returns a zero rate for the velocity property for the position component class.

<void> velocitySet(<double precision[:]> value) Set the velocity property of the position component.

nodeComponentSatellite

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<double precision> boundMass() Returns the default value for the boundMass property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> boundMassAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the boundMass property

<integer> boundMassCount() Compute the count of evolvable quantities in the boundMass property of the SatelliteOrbiting component.

<type(history)> boundMassHistory() Returns the default value for the boundMassHistory property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> boundMassHistoryAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the boundMassHistory property

<logical> boundMassHistoryIsGettable() Returns true if the boundMassHistory property is gettable for the satellite component class.

<logical> boundMassHistoryIsSettable() Specify whether the boundMassHistory property of the satellite component is settable.

<type(history)> boundMassHistoryRateGet() Returns a zero rate for the boundMassHistory property for the satellite component class.

<void> boundMassHistorySet(<type(history)> value) Set the boundMassHistory property of the satellite component.

<logical> boundMassIsGettable() Returns true if the boundMass property is gettable for the satellite component class.

<logical> boundMassIsSettable() Specify whether the boundMass property of the satellite component is settable.

<void> boundMassRate(<double precision> value) Cumulate to the rate of the boundMass property of the SatelliteOrbiting component.

<double precision> boundMassRateGet() Returns a zero rate for the boundMass property for the satellite component class.

<void> boundMassScale(<double precision> value) Set the scale of the boundMass property of the SatelliteOrbiting component.

<void> boundMassSet(<double precision> value) Set the boundMass property of the satellite component.

<void> builder(<*type(node)> componentDefinition→) Build a generic satellite component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Read properties from raw file.

<void> deserializeValues(<double(:)> array→, <integer> propertyType→) Deserialize the evolvable quantities from an array.

<void> destroy() Finalize a generic satellite component.

<void> dumpASCII() Dump the content of a satellite component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a generic satellite component.

<logical> isOrphan() Returns the default value for the isOrphan property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> isOrphanAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the isOrphan property

<logical> isOrphanIsGettable() Returns true if the isOrphan property is gettable for the satellite component class.

<logical> isOrphanIsSettable() Specify whether the isOrphan property of the satellite component is settable.

<logical> isOrphanRateGet() Returns a zero rate for the isOrphan property for the satellite component class.

<void> isOrphanSet(<logical> value) Set the isOrphan property of the satellite component.

<double precision> mergeTime() Returns the default value for the mergeTime property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> mergeTimeAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the mergeTime property

<integer> mergeTimeCount() Compute the count of evolvable quantities in the mergeTime property of the SatelliteStandard component.

<logical> mergeTimeIsGettable() Returns true if the mergeTime property is gettable for the satellite component class.

<logical> mergeTimeIsSettable() Specify whether the mergeTime property of the satellite component is settable.

<void> mergeTimeRate(<double precision> value) Cumulate to the rate of the mergeTime property of the SatelliteStandard component.

<double precision> mergeTimeRateGet() Returns a zero rate for the mergeTime property for the satellite component class.

<void> mergeTimeScale(<double precision> value) Set the scale of the mergeTime property of the SatelliteStandard component.

<void> mergeTimeSet(<double precision> value) Set the mergeTime property of the satellite component.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a nodeComponent object.

<integer(kind=kind_int8) => classDefault> nodeIndex() Returns the default value for the nodeIndex property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> nodeIndexAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the nodeIndex property

<type(longIntegerHistory)> nodeIndexHistory() Returns the default value for the nodeIndexHistory property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> nodeIndexHistoryAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the nodeIndexHistory property

<logical> nodeIndexHistoryIsGettable() Returns true if the nodeIndexHistory property is gettable for the satellite component class.

<logical> nodeIndexHistoryIsSettable() Specify whether the nodeIndexHistory property of the satellite component is settable.

<type(longIntegerHistory)> nodeIndexHistoryRateGet() Returns a zero rate for the nodeIndexHistory property for the satellite component class.

<void> nodeIndexHistorySet(<type(longIntegerHistory)> value) Set the nodeIndexHistory property of the satellite component.

<logical> nodeIndexIsGettable() Returns true if the nodeIndex property is gettable for the satellite component class.

<logical> nodeIndexIsSettable() Specify whether the nodeIndex property of the satellite component is settable.

<integer(kind=kind_int8) => classDefault> nodeIndexRateGet() Returns a zero rate for the nodeIndex property for the satellite component class.

<logical> nullIsActive() Return true if the null implementation of the satellite component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<logical> orbitingIsActive() Return true if the orbiting implementation of the satellite component is the active choice.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a satellite component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a generic satellite component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Establish the names of properties to output for a generic satellite component.

<double precision, dimension(:), allocatable => classDefault> position() Returns the default value for the position property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> positionAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the position property

<integer> positionCount() Compute the count of evolvable quantities in the position property of the SatelliteOrbiting component.

<logical> positionIsGettable() Returns true if the position property is gettable for the satellite component class.

<logical> positionIsSettable() Specify whether the position property of the satellite component is settable.

<void> positionRate(<double precision[:]> value) Cumulate to the rate of the position property of the SatelliteOrbiting component.

<double precision, dimension(:), allocatable => classDefault> positionRateGet() Returns a zero rate for the position property for the satellite component class.

<void> positionScale(<double precision[:]> value) Set the scale of the position property of the SatelliteOrbiting component.

<void> positionSet(<double precision[:]> value) Set the position property of the satellite component.

<void> postOutput(<double precision> time→) Perform post-output processing of a satellite component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<logical> presetIsActive() Return true if the preset implementation of the satellite component is the active choice.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets() Set offsets into serialization arrays.

<void> serializeASCII() Serialize the content of a satellite component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the number of evolvable quantities to be evolved.

<void> serializeRaw(<integer> fileHandle→) Generate a binary dump of all properties.

<void> serializeValues(<double(:)> array←, <integer> propertyType→) Serialize the evolvable quantities to an array.

<void> serializeXML() Generate an XML dump of all properties.

<integer(c_size_t)> sizeOf() Return the size in bytes of a nodeComponentSatellite component.

<logical> standardIsActive() Return true if the standard implementation of the satellite component is the active choice.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<double precision> tidalHeatingNormalized() Returns the default value for the tidalHeatingNormalized property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> tidalHeatingNormalizedAttributeMatch(<log [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the tidalHeatingNormalized property

<integer> tidalHeatingNormalizedCount() Compute the count of evolvable quantities in the tidalHeatingNormalized property of the SatelliteOrbiting component.

<logical> tidalHeatingNormalizedIsGettable() Returns true if the tidalHeatingNormalized property is gettable for the satellite component class.

<logical> tidalHeatingNormalizedIsSettable() Specify whether the tidalHeatingNormalized property of the satellite component is settable.

<void> tidalHeatingNormalizedRate(<double precision> value) Cumulate to the rate of the tidalHeatingNormalize property of the SatelliteOrbiting component.

<double precision> tidalHeatingNormalizedRateGet() Returns a zero rate for the tidalHeatingNormalized property for the satellite component class.

<void> tidalHeatingNormalizedScale(<double precision> value) Set the scale of the tidalHeatingNormalized property of the SatelliteOrbiting component.

<void> tidalHeatingNormalizedSet(<double precision> value) Set the tidalHeatingNormalized property of the satellite component.

<type(tensorRank2Dimension3Symmetric)> tidalTensorPathIntegrated() Returns the default value for the tidalTensorPathIntegrated property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> tidalTensorPathIntegratedAttributeMatch(<
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the satellite class that have the desired attributes for the tidalTensorPathIntegrated property

<integer> tidalTensorPathIntegratedCount() Compute the count of evolvable quantities in the tidalTensorPathIntegrated property of the SatelliteOrbiting component.

<logical> tidalTensorPathIntegratedIsGettable() Returns true if the tidalTensorPathIntegrated property is gettable for the satellite component class.

<logical> tidalTensorPathIntegratedIsSettable() Specify whether the tidalTensorPathIntegrated property of the satellite component is settable.

<void> tidalTensorPathIntegratedRate(<type(tensorRank2Dimension3Symmetric)> value) Cumulate to the rate of the tidalTensorPathIntegrated property of the SatelliteOrbiting component.

<type(tensorRank2Dimension3Symmetric)> tidalTensorPathIntegratedRateGet() Returns a zero rate for the tidalTensorPathIntegrated property for the satellite component class.

<void> tidalTensorPathIntegratedScale(<type(tensorRank2Dimension3Symmetric)> value) Set the scale of the tidalTensorPathIntegrated property of the SatelliteOrbiting component.

<void> tidalTensorPathIntegratedSet(<type(tensorRank2Dimension3Symmetric)> value) Set the tidalTensorPathIntegrated property of the satellite component.

<double precision> timeOfMerging() Returns the default value for the timeOfMerging property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> timeOfMergingAttributeMatch(<logical>
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the satellite class that have the desired attributes for the timeOfMerging property

<logical> timeOfMergingIsGettable() Returns true if the timeOfMerging property is gettable for the satellite component class.

<logical> timeOfMergingIsSettable() Specify whether the timeOfMerging property of the satellite component is settable.

<double precision> timeOfMergingRateGet() Returns a zero rate for the timeOfMerging property for the satellite component class.

<void> timeOfMergingSet(<double precision> value) Set the timeOfMerging property of the satellite component.

<type(varying_string)> type() Returns the type name for the satellite component class.

<double precision, dimension(:), allocatable => classDefault> velocity() Returns the default value for the velocity property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> velocityAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the velocity property

<integer> velocityCount() Compute the count of evolvable quantities in the velocity property of the SatelliteOrbiting component.

<logical> velocityIsGettable() Returns true if the velocity property is gettable for the satellite component class.

<logical> velocityIsSettable() Specify whether the velocity property of the satellite component is settable.

<void> velocityRate(<double precision[:]> value) Cumulate to the rate of the velocity property of the SatelliteOrbiting component.

<double precision, dimension(:), allocatable => classDefault> velocityRateGet() Returns a zero rate for the velocity property for the satellite component class.

<void> velocityScale(<double precision[:]> value) Set the scale of the velocity property of the SatelliteOrbiting component.

<void> velocitySet(<double precision[:]> value) Set the velocity property of the satellite component.

<logical> verySimpleIsActive() Return true if the verySimple implementation of the satellite component is the active choice.

<type(keplerOrbit)> virialOrbit() Returns the default value for the virialOrbit property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> virialOrbitAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the virialOrbit property

<logical> virialOrbitIsGettable() Returns true if the virialOrbit property is gettable for the satellite component class.

<logical> virialOrbitIsSettable() Specify whether the virialOrbit property of the satellite component is settable.

<type(keplerOrbit)> virialOrbitRateGet() Returns a zero rate for the virialOrbit property for the satellite component class.

<void> virialOrbitSet(<type(keplerOrbit)> value) Set the virialOrbit property of the satellite component.

nodeComponentSatelliteNull

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<double precision> boundMass() Returns the default value for the boundMass property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> boundMassAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the boundMass property

<integer> boundMassCount() Compute the count of evolvable quantities in the boundMass property of the SatelliteOrbiting component.

<type(history)> boundMassHistory() Returns the default value for the boundMassHistory property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> boundMassHistoryAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the boundMassHistory property

<logical> boundMassHistoryIsGettable() Returns true if the boundMassHistory property is gettable for the satellite component class.

<logical> boundMassHistoryIsSettable() Specify whether the boundMassHistory property of the satellite component is settable.

<type(history)> boundMassHistoryRateGet() Returns a zero rate for the boundMassHistory property for the satellite component class.

<void> boundMassHistorySet(<type(history)> value) Set the boundMassHistory property of the satellite component.

<logical> boundMassIsGettable() Returns true if the boundMass property is gettable for the satellite component class.

<logical> boundMassIsSettable() Specify whether the boundMass property of the satellite component is settable.

<void> boundMassRate(<double precision> value) Cumulate to the rate of the boundMass property of the SatelliteOrbiting component.

<double precision> boundMassRateGet() Returns a zero rate for the boundMass property for the satellite component class.

<void> boundMassScale(<double precision> value) Set the scale of the boundMass property of the SatelliteOrbiting component.

<void> boundMassSet(<double precision> value) Set the boundMass property of the satellite component.

<void> builder(<*type(node)> componentDefinition→) Build a null implementation of the satellite component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a null implementation of the satellite component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a null implementation of the satellite component from array.

<void> destroy() Finalize a null implementation of the satellite component.

<void> dumpASCII() Dump the content of a satellite component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a null member of the satellite component.

<logical> isOrphan() Returns the default value for the isOrphan property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> isOrphanAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the isOrphan property

<logical> isOrphanIsGettable() Returns true if the isOrphan property is gettable for the satellite component class.

<logical> isOrphanIsSettable() Specify whether the isOrphan property of the satellite component is settable.

<logical> isOrphanRateGet() Returns a zero rate for the isOrphan property for the satellite component class.

<void> isOrphanSet(<logical> value) Set the isOrphan property of the satellite component.

<double precision> mergeTime() Returns the default value for the mergeTime property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> mergeTimeAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the mergeTime property

<integer> mergeTimeCount() Compute the count of evolvable quantities in the mergeTime property of the SatelliteStandard component.

<logical> mergeTimeIsGettable() Returns true if the mergeTime property is gettable for the satellite component class.

<logical> mergeTimeIsSettable() Specify whether the mergeTime property of the satellite component is settable.

<void> mergeTimeRate(<double precision> value) Cumulate to the rate of the mergeTime property of the SatelliteStandard component.

<double precision> mergeTimeRateGet() Returns a zero rate for the mergeTime property for the satellite component class.

<void> mergeTimeScale(<double precision> value) Set the scale of the mergeTime property of the SatelliteStandard component.

<void> mergeTimeSet(<double precision> value) Set the mergeTime property of the satellite component.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a null implementation of the satellite component class.

<integer(kind=kind_int8) => classDefault> nodeIndex() Returns the default value for the nodeIndex property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> nodeIndexAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the nodeIndex property

<type(longIntegerHistory)> nodeIndexHistory() Returns the default value for the nodeIndexHistory property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> nodeIndexHistoryAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the nodeIndexHistory property

<logical> nodeIndexHistoryIsGettable() Returns true if the nodeIndexHistory property is gettable for the satellite component class.

<logical> nodeIndexHistoryIsSettable() Specify whether the nodeIndexHistory property of the satellite component is settable.

<type(longIntegerHistory)> nodeIndexHistoryRateGet() Returns a zero rate for the nodeIndexHistory property for the satellite component class.

<void> nodeIndexHistorySet(<type(longIntegerHistory)> value) Set the nodeIndexHistory property of the satellite component.

<logical> nodeIndexIsGettable() Returns true if the nodeIndex property is gettable for the satellite component class.

<logical> nodeIndexIsSettable() Specify whether the nodeIndex property of the satellite component is settable.

<integer(kind=kind_int8) => classDefault> nodeIndexRateGet() Returns a zero rate for the nodeIndex property for the satellite component class.

<logical> nullIsActive() Return true if the null implementation of the satellite component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<logical> orbitingIsActive() Return true if the orbiting implementation of the satellite component is the active choice.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a satellite component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a null implementation of the satellite component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a null implementation of the satellite component.

<double precision, dimension(:), allocatable => classDefault> position() Returns the default value for the position property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> positionAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the position property

`<integer> positionCount()` Compute the count of evolvable quantities in the `position` property of the `SatelliteOrbiting` component.

`<logical> positionIsGettable()` Returns true if the `position` property is gettable for the `satellite` component class.

`<logical> positionIsSettable()` Specify whether the `position` property of the `satellite` component is settable.

`<void> positionRate(<double precision[:]> value)` Cumulate to the rate of the `position` property of the `SatelliteOrbiting` component.

`<double precision, dimension(:), allocatable => classDefault> positionRateGet()` Returns a zero rate for the `position` property for the `satellite` component class.

`<void> positionScale(<double precision[:]> value)` Set the scale of the `position` property of the `SatelliteOrbiting` component.

`<void> positionSet(<double precision[:]> value)` Set the `position` property of the `satellite` component.

`<void> postOutput(<double precision> time→)` Perform post-output processing of a `satellite` component.

`<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→)` Compute the gravitational potential.

`<logical> presetIsActive()` Return true if the preset implementation of the satellite component is the active choice.

`<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→)` Compute the rotation curve.

`<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→)` Compute the rotation curve gradient.

`<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→)` Compute offsets into serialization arrays for a `null` implementation of the `satellite` component.

`<void> serializeASCII()` Serialize the contents of a null implementation of the satellite component to ASCII.

`<integer> serializeCount(<integer> propertyType→)` Return a count of the serialization of a null implementation of the satellite component.

`<void> serializeRaw(<integer> fileHandle→)` Serialize the contents of a null implementation of the satellite component to raw (binary) file.

`<void> serializeValues(<double precision[:]> array←, <integer> propertyType→)` Serialize evolvable properties of a `null` implementation of the `satellite` component to array.

`<void> serializeXML(<integer> fileHandle→)` Serialize the contents of a null implementation of the satellite component to XML.

`<integer(c_size_t)> sizeOf()` Return the size in bytes of a null implementation of the satellite component.

<logical> standardIsActive() Return true if the standard implementation of the satellite component is the active choice.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<double precision> tidalHeatingNormalized() Returns the default value for the tidalHeatingNormalized property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> tidalHeatingNormalizedAttributeMatch(<log [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the tidalHeatingNormalized property

<integer> tidalHeatingNormalizedCount() Compute the count of evolvable quantities in the tidalHeatingNormalized property of the SatelliteOrbiting component.

<logical> tidalHeatingNormalizedIsGettable() Returns true if the tidalHeatingNormalized property is gettable for the satellite component class.

<logical> tidalHeatingNormalizedIsSettable() Specify whether the tidalHeatingNormalized property of the satellite component is settable.

<void> tidalHeatingNormalizedRate(<double precision> value) Cumulate to the rate of the tidalHeatingNormalize property of the SatelliteOrbiting component.

<double precision> tidalHeatingNormalizedRateGet() Returns a zero rate for the tidalHeatingNormalized property for the satellite component class.

<void> tidalHeatingNormalizedScale(<double precision> value) Set the scale of the tidalHeatingNormalized property of the SatelliteOrbiting component.

<void> tidalHeatingNormalizedSet(<double precision> value) Set the tidalHeatingNormalized property of the satellite component.

<type(tensorRank2Dimension3Symmetric)> tidalTensorPathIntegrated() Returns the default value for the tidalTensorPathIntegrated property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> tidalTensorPathIntegratedAttributeMatch(< [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the tidalTensorPathIntegrated property

<integer> tidalTensorPathIntegratedCount() Compute the count of evolvable quantities in the tidalTensorPathIntegrated property of the SatelliteOrbiting component.

<logical> tidalTensorPathIntegratedIsGettable() Returns true if the tidalTensorPathIntegrated property is gettable for the satellite component class.

<logical> tidalTensorPathIntegratedIsSettable() Specify whether the tidalTensorPathIntegrated property of the satellite component is settable.

<void> tidalTensorPathIntegratedRate(<type(tensorRank2Dimension3Symmetric)> value) Cumulate to the rate of the tidalTensorPathIntegrated property of the SatelliteOrbiting component.

<type(tensorRank2Dimension3Symmetric)> tidalTensorPathIntegratedRateGet() Returns a zero rate for the tidalTensorPathIntegrated property for the satellite component class.

<void> tidalTensorPathIntegratedScale(<type(tensorRank2Dimension3Symmetric)> value) Set the scale of the tidalTensorPathIntegrated property of the SatelliteOrbiting component.

<void> tidalTensorPathIntegratedSet(<type(tensorRank2Dimension3Symmetric)> value) Set the tidalTensorPathIntegrated property of the satellite component.

<double precision> timeOfMerging() Returns the default value for the timeOfMerging property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> timeOfMergingAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the timeOfMerging property

<logical> timeOfMergingIsGettable() Returns true if the timeOfMerging property is gettable for the satellite component class.

<logical> timeOfMergingIsSettable() Specify whether the timeOfMerging property of the satellite component is settable.

<double precision> timeOfMergingRateGet() Returns a zero rate for the timeOfMerging property for the satellite component class.

<void> timeOfMergingSet(<double precision> value) Set the timeOfMerging property of the satellite component.

<type(varying_string)> type() Returns the type name for the null implementation of the satellite component class.

<double precision, dimension(:), allocatable => classDefault> velocity() Returns the default value for the velocity property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> velocityAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the velocity property

<integer> velocityCount() Compute the count of evolvable quantities in the velocity property of the SatelliteOrbiting component.

<logical> velocityIsGettable() Returns true if the velocity property is gettable for the satellite component class.

<logical> velocityIsSettable() Specify whether the velocity property of the satellite component is settable.

<void> velocityRate(<double precision[:]> value) Cumulate to the rate of the velocity property of the SatelliteOrbiting component.

<double precision, dimension(:), allocatable => classDefault> velocityRateGet() Returns a zero rate for the velocity property for the satellite component class.

<void> velocityScale(<double precision[:]> value) Set the scale of the velocity property of the SatelliteOrbiting component.

<void> velocitySet(<double precision[:]> value) Set the velocity property of the satellite component.

<logical> verySimpleIsActive() Return true if the verySimple implementation of the satellite component is the active choice.

<type(keplerOrbit)> virialOrbit() Returns the default value for the virialOrbit property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> virialOrbitAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the virialOrbit property

<logical> virialOrbitIsGettable() Returns true if the virialOrbit property is gettable for the satellite component class.

<logical> virialOrbitIsSettable() Specify whether the virialOrbit property of the satellite component is settable.

<type(keplerOrbit)> virialOrbitRateGet() Returns a zero rate for the virialOrbit property for the satellite component class.

<void> virialOrbitSet(<type(keplerOrbit)> value) Set the virialOrbit property of the satellite component.

nodeComponentSatelliteOrbiting

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<double precision> boundMass() Get the boundMass property of an orbiting implementation of the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> boundMassAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the boundMass property

<integer> boundMassCount() Return a count of the number of scalar properties in the boundMass property of an orbiting implementation of the satellite component class.

<type(history)> boundMassHistory() Returns the default value for the boundMassHistory property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> boundMassHistoryAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the boundMassHistory property

<logical> boundMassHistoryIsGettable() Returns true if the boundMassHistory property is gettable for the satellite component class.

<logical> boundMassHistoryIsSettable() Specify whether the boundMassHistory property of the satellite component is settable.

<type(history)> boundMassHistoryRateGet() Returns a zero rate for the boundMassHistory property for the satellite component class.

<void> boundMassHistorySet(<type(history)> value) Set the boundMassHistory property of the satellite component.

<void> boundMassInactive() Indicate that the boundMass property of an orbiting implementation of the satellite component class is inactive for differential equation solving.

<logical> boundMassIsGettable() Returns true if the boundMass property is gettable for the satellite component class.

<logical> boundMassIsSettable() Specify whether the boundMass property of the satellite component is settable.

<void> boundMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the boundMass property of an orbiting implementation of the satellite component class.

<double precision> boundMassRateGet() Get the rate of change of the boundMass property of an orbiting implementation of the satellite component class.

<void> boundMassScale(<double precision> setValue→) Set the absolute scale of the boundMass property of an orbiting implementation of the satellite component class.

<void> boundMassSet(<double precision> setValue→) Set the boundMass property of an orbiting implementation of the satellite component class.

<void> builder(<*type(node)> componentDefinition→) Build a orbiting implementation of the satellite component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a orbiting implementation of the satellite component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a orbiting implementation of the satellite component from array.

<void> destroy() Finalize a orbiting implementation of the satellite component.

<void> dumpASCII() Dump the content of a satellite component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a orbiting member of the satellite component.

<logical> isOrphan() Returns the default value for the isOrphan property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> isOrphanAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the isOrphan property

<logical> isOrphanIsGettable() Returns true if the isOrphan property is gettable for the satellite component class.

<logical> isOrphanIsSettable() Specify whether the isOrphan property of the satellite component is settable.

<logical> isOrphanRateGet() Returns a zero rate for the isOrphan property for the satellite component class.

<void> isOrphanSet(<logical> value) Set the isOrphan property of the satellite component.

<double precision> mergeTime() Get the mergeTime property of an orbiting implementation of the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> mergeTimeAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the mergeTime property

<integer> mergeTimeCount() Compute the count of evolvable quantities in the mergeTime property of the SatelliteStandard component.

<logical> mergeTimeIsGettable() Returns true if the mergeTime property is gettable for the satellite component class.

<logical> mergeTimeIsSettable() Specify whether the mergeTime property of the satellite component is settable.

<void> mergeTimeRate(<double precision> value) Cumulate to the rate of the mergeTime property of the SatelliteStandard component.

<double precision> mergeTimeRateGet() Returns a zero rate for the mergeTime property for the satellite component class.

<void> mergeTimeScale(<double precision> value) Set the scale of the mergeTime property of the SatelliteStandard component.

<void> mergeTimeSet(<double precision> setValue→) Set the mergeTime property of an orbiting implementation of the satellite component class.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a orbiting implementation of the satellite component class.

<integer(kind=kind_int8) => classDefault> nodeIndex() Returns the default value for the nodeIndex property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> nodeIndexAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the nodeIndex property

<type(longIntegerHistory)> nodeIndexHistory() Returns the default value for the nodeIndexHistory property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> nodeIndexHistoryAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the nodeIndexHistory property

<logical> nodeIndexHistoryIsGettable() Returns true if the nodeIndexHistory property is gettable for the satellite component class.

<logical> nodeIndexHistoryIsSettable() Specify whether the nodeIndexHistory property of the satellite component is settable.

<type(longIntegerHistory)> nodeIndexHistoryRateGet() Returns a zero rate for the nodeIndexHistory property for the satellite component class.

<void> nodeIndexHistorySet(<type(longIntegerHistory)> value) Set the nodeIndexHistory property of the satellite component.

<logical> nodeIndexIsGettable() Returns true if the nodeIndex property is gettable for the satellite component class.

<logical> nodeIndexIsSettable() Specify whether the nodeIndex property of the satellite component is settable.

<integer(kind=kind_int8) => classDefault> nodeIndexRateGet() Returns a zero rate for the nodeIndex property for the satellite component class.

<logical> nullIsActive() Return true if the null implementation of the satellite component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<logical> orbitingIsActive() Return true if the orbiting implementation of the satellite component is the active choice.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_-int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a satellite component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a orbiting implementation of the satellite component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a orbiting implementation of the satellite component.

<double precision, dimension(:), allocatable => propertyValue> position() Get the position property of an orbiting implementation of the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> positionAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the position property

<integer> positionCount() Return a count of the number of scalar properties in the position property of an orbiting implementation of the satellite component class.

<void> positionInactive() Indicate that the position property of an orbiting implementation of the satellite component class is inactive for differential equation solving.

<logical> positionIsGettable() Returns true if the position property is gettable for the satellite component class.

<logical> positionIsSettable() Specify whether the position property of the satellite component is settable.

<void> positionRate(<double precision[:]> setValue→, <logical> [interrupt]↔, <*procedure(interruptTas [interruptProcedure]↔) Accumulate to the rate of change of the position property of an orbiting implementation of the satellite component class.

<double precision, dimension(:), allocatable => propertyRate> positionRateGet() Get the rate of change of the position property of an orbiting implementation of the satellite component class.

<void> positionScale(<double precision[:]> setValue→) Set the absolute scale of the position property of an orbiting implementation of the satellite component class.

<void> positionSet(<double precision[:]> setValue→) Set the position property of an orbiting implementation of the satellite component class.

<void> postOutput(<double precision> time→) Perform post-output processing of a satellite component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<logical> presetIsActive() Return true if the preset implementation of the satellite component is the active choice.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a orbiting implementation of the satellite component.

<void> serializeASCII() Serialize the contents of a orbiting implementation of the satellite component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a orbiting implementation of the satellite component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a orbiting implementation of the satellite component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a orbiting implementation of the satellite component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a orbiting implementation of the satellite component to XML.

<integer(c_size_t)> sizeOf() Return the size in bytes of a orbiting implementation of the satellite component.

<logical> standardIsActive() Return true if the standard implementation of the satellite component is the active choice.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<double precision> tidalHeatingNormalized() Get the tidalHeatingNormalized property of an orbiting implementation of the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> tidalHeatingNormalizedAttributeMatch(<log [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the tidalHeatingNormalized property

<integer> tidalHeatingNormalizedCount() Return a count of the number of scalar properties in the tidalHeatingNormalized property of an orbiting implementation of the satellite component class.

<void> tidalHeatingNormalizedInactive() Indicate that the tidalHeatingNormalized property of an orbiting implementation of the satellite component class is inactive for differential equation solving.

<logical> tidalHeatingNormalizedIsGettable() Returns true if the tidalHeatingNormalized property is gettable for the satellite component class.

<logical> tidalHeatingNormalizedIsSettable() Specify whether the tidalHeatingNormalized property of the satellite component is settable.

<void> tidalHeatingNormalizedRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the tidalHeatingNormalized property of an orbiting implementation of the satellite component class.

<double precision> tidalHeatingNormalizedRateGet() Get the rate of change of the tidalHeatingNormalized property of an orbiting implementation of the satellite component class.

<void> tidalHeatingNormalizedScale(<double precision> setValue→) Set the absolute scale of the tidalHeatingNormalized property of an orbiting implementation of the satellite component class.

<void> tidalHeatingNormalizedSet(<double precision> setValue→) Set the tidalHeatingNormalized property of an orbiting implementation of the satellite component class.

<type(tensorRank2Dimension3Symmetric)> tidalTensorPathIntegrated() Get the tidalTensorPathIntegrated property of an orbiting implementation of the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> tidalTensorPathIntegratedAttributeMatch(<
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the satellite class that have the desired attributes for the tidalTensorPathIntegrated property

<integer> tidalTensorPathIntegratedCount() Return a count of the number of scalar properties in the tidalTensorPathIntegrated property of an orbiting implementation of the satellite component class.

<void> tidalTensorPathIntegratedInactive() Indicate that the tidalTensorPathIntegrated property of an orbiting implementation of the satellite component class is inactive for differential equation solving.

<logical> tidalTensorPathIntegratedIsGettable() Returns true if the tidalTensorPathIntegrated property is gettable for the satellite component class.

<logical> tidalTensorPathIntegratedIsSettable() Specify whether the tidalTensorPathIntegrated property of the satellite component is settable.

<void> tidalTensorPathIntegratedRate(<type(tensorRank2Dimension3Symmetric)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the tidalTensorPathIntegrated property of an orbiting implementation of the satellite component class.

<type(tensorRank2Dimension3Symmetric)> tidalTensorPathIntegratedRateGet() Get the rate of change of the tidalTensorPathIntegrated property of an orbiting implementation of the satellite component class.

<void> tidalTensorPathIntegratedScale(<type(tensorRank2Dimension3Symmetric)> setValue→) Set the absolute scale of the tidalTensorPathIntegrated property of an orbiting implementation of the satellite component class.

<void> tidalTensorPathIntegratedSet(<type(tensorRank2Dimension3Symmetric)> setValue→) Set the tidalTensorPathIntegrated property of an orbiting implementation of the satellite component class.

<double precision> timeOfMerging() Returns the default value for the timeOfMerging property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> timeOfMergingAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the timeOfMerging property

<logical> timeOfMergingIsGettable() Returns true if the timeOfMerging property is gettable for the satellite component class.

<logical> timeOfMergingIsSettable() Specify whether the timeOfMerging property of the satellite component is settable.

<double precision> timeOfMergingRateGet() Returns a zero rate for the timeOfMerging property for the satellite component class.

<void> timeOfMergingSet(<double precision> value) Set the timeOfMerging property of the satellite component.

<type(varying_string)> type() Returns the type name for the orbiting implementation of the satellite component class.

<double precision, dimension(:), allocatable => propertyValue> velocity() Get the velocity property of an orbiting implementation of the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> velocityAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the velocity property

<integer> velocityCount() Return a count of the number of scalar properties in the velocity property of an orbiting implementation of the satellite component class.

<void> velocityInactive() Indicate that the velocity property of an orbiting implementation of the satellite component class is inactive for differential equation solving.

<logical> velocityIsGettable() Returns true if the velocity property is gettable for the satellite component class.

<logical> velocityIsSettable() Specify whether the velocity property of the satellite component is settable.

<void> velocityRate(<double precision[:]> setValue→, <logical> [interrupt]↔, <*procedure(interruptTas [interruptProcedure]↔) Accumulate to the rate of change of the velocity property of an orbiting implementation of the satellite component class.

<double precision, dimension(:), allocatable => propertyRate> velocityRateGet() Get the rate of change of the velocity property of an orbiting implementation of the satellite component class.

<void> velocityScale(<double precision[:]> setValue→) Set the absolute scale of the velocity property of an orbiting implementation of the satellite component class.

<void> velocitySet(<double precision[:]> setValue→) Set the velocity property of an orbiting implementation of the satellite component class.

<logical> verySimpleIsActive() Return true if the verySimple implementation of the satellite component is the active choice.

<type(keplerOrbit)> virialOrbit() Get the virialOrbit property of an orbiting implementation of the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> virialOrbitAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the virialOrbit property

<logical> virialOrbitIsGettable() Returns true if the virialOrbit property is gettable for the satellite component class.

<logical> virialOrbitIsSettable() Specify whether the virialOrbit property of the satellite component is settable.

<type(keplerOrbit)> virialOrbitRateGet() Returns a zero rate for the virialOrbit property for the satellite component class.

<void> virialOrbitSet(<type(keplerOrbit)> setValue→) Set the value of the virialOrbit property of the orbiting implementation of the satellite component using a deferred function.

<void> virialOrbitSetFunction() Set the function to be used for the set method of the virialOrbit property of the SatelliteOrbiting component.

<logical> virialOrbitSetIsAttached() Return true if the deferred function used to set the virialOrbit property of the SatelliteOrbiting component class has been attached.

<void> virialOrbitSetValue(<type(keplerOrbit)> setValue→) Set the virialOrbit property of an orbiting implementation of the satellite component class.

nodeComponentSatellitePreset

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<double precision> boundMass() Returns the default value for the boundMass property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> boundMassAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the boundMass property

<integer> boundMassCount() Compute the count of evolvable quantities in the boundMass property of the SatelliteOrbiting component.

<type(history)> boundMassHistory() Get the boundMassHistory property of an preset implementation of the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> boundMassHistoryAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the boundMassHistory property

<logical> boundMassHistoryIsGettable() Returns true if the boundMassHistory property is gettable for the satellite component class.

<logical> boundMassHistoryIsSettable() Specify whether the boundMassHistory property of the satellite component is settable.

<type(history)> boundMassHistoryRateGet() Returns a zero rate for the boundMassHistory property for the satellite component class.

<void> boundMassHistorySet(<type(history)> setValue→) Set the boundMassHistory property of an preset implementation of the satellite component class.

<logical> boundMassIsGettable() Returns true if the boundMass property is gettable for the satellite component class.

<logical> boundMassIsSettable() Specify whether the boundMass property of the satellite component is settable.

<void> boundMassRate(<double precision> value) Cumulate to the rate of the boundMass property of the SatelliteOrbiting component.

<double precision> boundMassRateGet() Returns a zero rate for the boundMass property for the satellite component class.

<void> boundMassScale(<double precision> value) Set the scale of the boundMass property of the SatelliteOrbiting component.

<void> boundMassSet(<double precision> value) Set the boundMass property of the satellite component.

<void> builder(<*type(node)> componentDefinition→) Build a preset implementation of the satellite component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a preset implementation of the satellite component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a preset implementation of the satellite component from array.

<void> destroy() Finalize a preset implementation of the satellite component.

<void> dumpASCII() Dump the content of a satellite component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a preset member of the satellite component.

<logical> isOrphan() Get the isOrphan property of an preset implementation of the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> isOrphanAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the isOrphan property

<logical> isOrphanIsGettable() Returns true if the isOrphan property is gettable for the satellite component class.

<logical> isOrphanIsSettable() Specify whether the isOrphan property of the satellite component is settable.

<logical> isOrphanRateGet() Returns a zero rate for the isOrphan property for the satellite component class.

<void> isOrphanSet(<logical> setValue→) Set the isOrphan property of an preset implementation of the satellite component class.

<double precision> mergeTime() Returns the default value for the mergeTime property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> mergeTimeAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the mergeTime property

<integer> mergeTimeCount() Compute the count of evolvable quantities in the mergeTime property of the SatelliteStandard component.

<logical> mergeTimeIsGettable() Returns true if the mergeTime property is gettable for the satellite component class.

<logical> mergeTimeIsSettable() Specify whether the mergeTime property of the satellite component is settable.

<void> mergeTimeRate(<double precision> value) Cumulate to the rate of the mergeTime property of the SatelliteStandard component.

<double precision> mergeTimeRateGet() Returns a zero rate for the mergeTime property for the satellite component class.

<void> mergeTimeScale(<double precision> value) Set the scale of the mergeTime property of the SatelliteStandard component.

<void> mergeTimeSet(<double precision> value) Set the mergeTime property of the satellite component.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a preset implementation of the satellite component class.

<integer(kind=kind_int8) => classDefault> nodeIndex() Returns the default value for the nodeIndex property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> nodeIndexAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the nodeIndex property

<type(longIntegerHistory)> nodeIndexHistory() Get the nodeIndexHistory property of an preset implementation of the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> nodeIndexHistoryAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the nodeIndexHistory property

<logical> nodeIndexHistoryIsGettable() Returns true if the nodeIndexHistory property is gettable for the satellite component class.

<logical> nodeIndexHistoryIsSettable() Specify whether the nodeIndexHistory property of the satellite component is settable.

<type(longIntegerHistory)> nodeIndexHistoryRateGet() Returns a zero rate for the nodeIndexHistory property for the satellite component class.

<void> nodeIndexHistorySet(<type(longIntegerHistory)> setValue→) Set the nodeIndexHistory property of an preset implementation of the satellite component class.

<logical> nodeIndexIsGettable() Returns true if the nodeIndex property is gettable for the satellite component class.

<logical> nodeIndexIsSettable() Specify whether the nodeIndex property of the satellite component is settable.

<integer(kind=kind_int8) => classDefault> nodeIndexRateGet() Returns a zero rate for the nodeIndex property for the satellite component class.

<logical> nullIsActive() Return true if the null implementation of the satellite component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<logical> orbitingIsActive() Return true if the orbiting implementation of the satellite component is the active choice.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_-int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a satellite component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a preset implementation of the satellite component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a preset implementation of the satellite component.

<double precision, dimension(:), allocatable => classDefault> position() Returns the default value for the position property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> positionAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the position property

<integer> positionCount() Compute the count of evolvable quantities in the position property of the SatelliteOrbiting component.

<logical> positionIsGettable() Returns true if the position property is gettable for the satellite component class.

<logical> positionIsSettable() Specify whether the position property of the satellite component is settable.

<void> positionRate(<double precision[:]> value) Cumulate to the rate of the position property of the SatelliteOrbiting component.

<double precision, dimension(:), allocatable => classDefault> positionRateGet() Returns a zero rate for the position property for the satellite component class.

<void> positionScale(<double precision[:]> value) Set the scale of the position property of the SatelliteOrbiting component.

<void> positionSet(<double precision[:]> value) Set the position property of the satellite component.

<void> postOutput(<double precision> time→) Perform post-output processing of a satellite component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<logical> presetIsActive() Return true if the preset implementation of the satellite component is the active choice.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a preset implementation of the satellite component.

<void> serializeASCII() Serialize the contents of a preset implementation of the satellite component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a preset implementation of the satellite component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a preset implementation of the satellite component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a preset implementation of the satellite component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a preset implementation of the satellite component to XML.

<integer(c_size_t)> sizeOf() Return the size in bytes of a preset implementation of the satellite component.

<logical> standardIsActive() Return true if the standard implementation of the satellite component is the active choice.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<double precision> tidalHeatingNormalized() Returns the default value for the tidalHeatingNormalized property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> tidalHeatingNormalizedAttributeMatch(<log [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the tidalHeatingNormalized property

<integer> tidalHeatingNormalizedCount() Compute the count of evolvable quantities in the tidalHeatingNormalized property of the SatelliteOrbiting component.

<logical> tidalHeatingNormalizedIsGettable() Returns true if the tidalHeatingNormalized property is gettable for the satellite component class.

<logical> tidalHeatingNormalizedIsSettable() Specify whether the tidalHeatingNormalized property of the satellite component is settable.

<void> tidalHeatingNormalizedRate(<double precision> value) Cumulate to the rate of the tidalHeatingNormalize property of the SatelliteOrbiting component.

<double precision> tidalHeatingNormalizedRateGet() Returns a zero rate for the tidalHeatingNormalized property for the satellite component class.

<void> tidalHeatingNormalizedScale(<double precision> value) Set the scale of the tidalHeatingNormalized property of the SatelliteOrbiting component.

<void> tidalHeatingNormalizedSet(<double precision> value) Set the tidalHeatingNormalized property of the satellite component.

<type(tensorRank2Dimension3Symmetric)> tidalTensorPathIntegrated() Returns the default value for the tidalTensorPathIntegrated property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> tidalTensorPathIntegratedAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the tidalTensorPathIntegrated property

<integer> tidalTensorPathIntegratedCount() Compute the count of evolvable quantities in the tidalTensorPathIntegrated property of the SatelliteOrbiting component.

<logical> tidalTensorPathIntegratedIsGettable() Returns true if the tidalTensorPathIntegrated property is gettable for the satellite component class.

<logical> tidalTensorPathIntegratedIsSettable() Specify whether the tidalTensorPathIntegrated property of the satellite component is settable.

<void> tidalTensorPathIntegratedRate(<type(tensorRank2Dimension3Symmetric)> value) Cumulate to the rate of the tidalTensorPathIntegrated property of the SatelliteOrbiting component.

<type(tensorRank2Dimension3Symmetric)> tidalTensorPathIntegratedRateGet() Returns a zero rate for the tidalTensorPathIntegrated property for the satellite component class.

<void> tidalTensorPathIntegratedScale(<type(tensorRank2Dimension3Symmetric)> value) Set the scale of the tidalTensorPathIntegrated property of the SatelliteOrbiting component.

<void> tidalTensorPathIntegratedSet(<type(tensorRank2Dimension3Symmetric)> value) Set the tidalTensorPathIntegrated property of the satellite component.

<double precision> timeOfMerging() Get the timeOfMerging property of an preset implementation of the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> timeOfMergingAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the timeOfMerging property

<logical> timeOfMergingIsGettable() Returns true if the timeOfMerging property is gettable for the satellite component class.

<logical> timeOfMergingIsSettable() Specify whether the timeOfMerging property of the satellite component is settable.

<double precision> timeOfMergingRateGet() Returns a zero rate for the timeOfMerging property for the satellite component class.

<void> timeOfMergingSet(<double precision> setValue→) Set the timeOfMerging property of an preset implementation of the satellite component class.

<type(varying_string)> type() Returns the type name for the preset implementation of the satellite component class.

<double precision, dimension(:), allocatable => classDefault> velocity() Returns the default value for the velocity property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> velocityAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the velocity property

<integer> velocityCount() Compute the count of evolvable quantities in the velocity property of the SatelliteOrbiting component.

<logical> velocityIsGettable() Returns true if the velocity property is gettable for the satellite component class.

<logical> velocityIsSettable() Specify whether the velocity property of the satellite component is settable.

<void> velocityRate(<double precision[:]> value) Cumulate to the rate of the velocity property of the SatelliteOrbiting component.

<double precision, dimension(:), allocatable => classDefault> velocityRateGet() Returns a zero rate for the velocity property for the satellite component class.

<void> velocityScale(<double precision[:]> value) Set the scale of the velocity property of the SatelliteOrbiting component.

<void> velocitySet(<double precision[:]> value) Set the velocity property of the satellite component.

<logical> verySimpleIsActive() Return true if the verySimple implementation of the satellite component is the active choice.

<type(keplerOrbit)> virialOrbit() Get the virialOrbit property of an preset implementation of the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> virialOrbitAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the virialOrbit property

<logical> virialOrbitIsGettable() Returns true if the virialOrbit property is gettable for the satellite component class.

<logical> virialOrbitIsSettable() Specify whether the virialOrbit property of the satellite component is settable.

<type(keplerOrbit)> virialOrbitRateGet() Returns a zero rate for the virialOrbit property for the satellite component class.

<void> virialOrbitSet(<type(keplerOrbit)> setValue→) Set the virialOrbit property of an preset implementation of the satellite component class.

nodeComponentSatelliteStandard

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<double precision> boundMass() Get the boundMass property of an standard implementation of the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> boundMassAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the boundMass property

<integer> boundMassCount() Return a count of the number of scalar properties in the boundMass property of an standard implementation of the satellite component class.

<type(history)> boundMassHistory() Returns the default value for the boundMassHistory property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> boundMassHistoryAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the boundMassHistory property

<logical> boundMassHistoryIsGettable() Returns true if the boundMassHistory property is gettable for the satellite component class.

<logical> boundMassHistoryIsSettable() Specify whether the boundMassHistory property of the satellite component is settable.

<type(history)> boundMassHistoryRateGet() Returns a zero rate for the boundMassHistory property for the satellite component class.

<void> boundMassHistorySet(<type(history)> value) Set the boundMassHistory property of the satellite component.

<void> boundMassInactive() Indicate that the boundMass property of an standard implementation of the satellite component class is inactive for differential equation solving.

<logical> boundMassIsGettable() Returns true if the boundMass property is gettable for the satellite component class.

<logical> boundMassIsSettable() Specify whether the boundMass property of the satellite component is settable.

<void> boundMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask) [interruptProcedure]↔) Accumulate to the rate of change of the boundMass property of an standard implementation of the satellite component class.

<double precision> boundMassRateGet() Get the rate of change of the boundMass property of an standard implementation of the satellite component class.

<void> boundMassScale(<double precision> setValue→) Set the absolute scale of the boundMass property of an standard implementation of the satellite component class.

<void> boundMassSet(<double precision> setValue→) Set the boundMass property of an standard implementation of the satellite component class.

<void> builder(<*type(node)> componentDefinition→) Build a standard implementation of the satellite component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a standard implementation of the satellite component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a standard implementation of the satellite component from array.

<void> destroy() Finalize a standard implementation of the satellite component.

<void> dumpASCII() Dump the content of a satellite component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a standard member of the satellite component.

<logical> isOrphan() Returns the default value for the isOrphan property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> isOrphanAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the isOrphan property

<logical> isOrphanIsGettable() Returns true if the isOrphan property is gettable for the satellite component class.

<logical> isOrphanIsSettable() Specify whether the isOrphan property of the satellite component is settable.

<logical> isOrphanRateGet() Returns a zero rate for the isOrphan property for the satellite component class.

<void> isOrphanSet(<logical> value) Set the isOrphan property of the satellite component.

<double precision> mergeTime() Returns the default value for the mergeTime property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> mergeTimeAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the mergeTime property

<integer> mergeTimeCount() Return a count of the number of scalar properties in the mergeTime property of an standard implementation of the satellite component class.

<void> mergeTimeInactive() Indicate that the mergeTime property of an standard implementation of the satellite component class is inactive for differential equation solving.

<logical> mergeTimeIsGettable() Returns true if the mergeTime property is gettable for the satellite component class.

<logical> mergeTimeIsSettable() Specify whether the mergeTime property of the satellite component is settable.

<void> mergeTimeRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask) [interruptProcedure]↔) Accumulate to the rate of change of the mergeTime property of an standard implementation of the satellite component class.

<double precision> mergeTimeRateGet() Get the rate of change of the mergeTime property of an standard implementation of the satellite component class.

<void> mergeTimeScale(<double precision> setValue→) Set the absolute scale of the mergeTime property of an standard implementation of the satellite component class.

<void> mergeTimeSet(<double precision> setValue→) Set the mergeTime property of an standard implementation of the satellite component class.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a standard implementation of the satellite component class.

<integer(kind=kind_int8) => classDefault> nodeIndex() Returns the default value for the nodeIndex property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> nodeIndexAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the nodeIndex property

<type(longIntegerHistory)> nodeIndexHistory() Returns the default value for the nodeIndexHistory property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> nodeIndexHistoryAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the nodeIndexHistory property

<logical> nodeIndexHistoryIsGettable() Returns true if the nodeIndexHistory property is gettable for the satellite component class.

<logical> nodeIndexHistoryIsSettable() Specify whether the nodeIndexHistory property of the satellite component is settable.

<type(longIntegerHistory)> nodeIndexHistoryRateGet() Returns a zero rate for the nodeIndexHistory property for the satellite component class.

<void> nodeIndexHistorySet(<type(longIntegerHistory)> value) Set the nodeIndexHistory property of the satellite component.

<logical> nodeIndexIsGettable() Returns true if the nodeIndex property is gettable for the satellite component class.

<logical> nodeIndexIsSettable() Specify whether the nodeIndex property of the satellite component is settable.

<integer(kind=kind_int8) => classDefault> nodeIndexRateGet() Returns a zero rate for the nodeIndex property for the satellite component class.

<logical> nullIsActive() Return true if the null implementation of the satellite component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<logical> orbitingIsActive() Return true if the orbiting implementation of the satellite component is the active choice.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a satellite component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a standard implementation of the satellite component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a standard implementation of the satellite component.

<double precision, dimension(:), allocatable => classDefault> position() Returns the default value for the position property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> positionAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the position property

<integer> positionCount() Compute the count of evolvable quantities in the position property of the SatelliteOrbiting component.

<logical> positionIsGettable() Returns true if the position property is gettable for the satellite component class.

<logical> positionIsSettable() Specify whether the position property of the satellite component is settable.

<void> positionRate(<double precision[:]> value) Cumulate to the rate of the position property of the SatelliteOrbiting component.

<double precision, dimension(:), allocatable => classDefault> positionRateGet() Returns a zero rate for the position property for the satellite component class.

<void> positionScale(<double precision[:]> value) Set the scale of the position property of the SatelliteOrbiting component.

<void> positionSet(<double precision[:]> value) Set the position property of the satellite component.

<void> postOutput(<double precision> time→) Perform post-output processing for a standard implementation of the satellite component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<logical> presetIsActive() Return true if the preset implementation of the satellite component is the active choice.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a standard implementation of the satellite component.

<void> serializeASCII() Serialize the contents of a standard implementation of the satellite component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a standard implementation of the satellite component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a standard implementation of the satellite component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a standard implementation of the satellite component to array.

`<void> serializeXML(<integer> fileHandle→)` Serialize the contents of a standard implementation of the satellite component to XML.

`<integer(c_size_t)> sizeOf()` Return the size in bytes of a standard implementation of the satellite component.

`<logical> standardIsActive()` Return true if the standard implementation of the satellite component is the active choice.

`<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→)` Compute the surface density.

`<double precision> tidalHeatingNormalized()` Returns the default value for the `tidalHeatingNormalized` property for the `satellite` component class.

`<type(varying_string), allocatable, dimension(:) => matches> tidalHeatingNormalizedAttributeMatch(<log [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)` Return a text list of component implementations in the `satellite` class that have the desired attributes for the `tidalHeatingNormalized` property

`<integer> tidalHeatingNormalizedCount()` Compute the count of evolvable quantities in the `tidalHeatingNormalized` property of the `SatelliteOrbiting` component.

`<logical> tidalHeatingNormalizedIsGettable()` Returns true if the `tidalHeatingNormalized` property is gettable for the `satellite` component class.

`<logical> tidalHeatingNormalizedIsSettable()` Specify whether the `tidalHeatingNormalized` property of the `satellite` component is settable.

`<void> tidalHeatingNormalizedRate(<double precision> value)` Cumulate to the rate of the `tidalHeatingNormalize` property of the `SatelliteOrbiting` component.

`<double precision> tidalHeatingNormalizedRateGet()` Returns a zero rate for the `tidalHeatingNormalized` property for the `satellite` component class.

`<void> tidalHeatingNormalizedScale(<double precision> value)` Set the scale of the `tidalHeatingNormalized` property of the `SatelliteOrbiting` component.

`<void> tidalHeatingNormalizedSet(<double precision> value)` Set the `tidalHeatingNormalized` property of the `satellite` component.

`<type(tensorRank2Dimension3Symmetric)> tidalTensorPathIntegrated()` Returns the default value for the `tidalTensorPathIntegrated` property for the `satellite` component class.

`<type(varying_string), allocatable, dimension(:) => matches> tidalTensorPathIntegratedAttributeMatch(< [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)` Return a text list of component implementations in the `satellite` class that have the desired attributes for the `tidalTensorPathIntegrated` property

`<integer> tidalTensorPathIntegratedCount()` Compute the count of evolvable quantities in the `tidalTensorPathIntegrated` property of the `SatelliteOrbiting` component.

`<logical> tidalTensorPathIntegratedIsGettable()` Returns true if the `tidalTensorPathIntegrated` property is gettable for the `satellite` component class.

<logical> tidalTensorPathIntegratedIsSettable() Specify whether the tidalTensorPathIntegrated property of the satellite component is settable.

<void> tidalTensorPathIntegratedRate(<type(tensorRank2Dimension3Symmetric)> value) Cumulate to the rate of the tidalTensorPathIntegrated property of the SatelliteOrbiting component.

<type(tensorRank2Dimension3Symmetric)> tidalTensorPathIntegratedRateGet() Returns a zero rate for the tidalTensorPathIntegrated property for the satellite component class.

<void> tidalTensorPathIntegratedScale(<type(tensorRank2Dimension3Symmetric)> value) Set the scale of the tidalTensorPathIntegrated property of the SatelliteOrbiting component.

<void> tidalTensorPathIntegratedSet(<type(tensorRank2Dimension3Symmetric)> value) Set the tidalTensorPathIntegrated property of the satellite component.

<double precision> timeOfMerging() Returns the default value for the timeOfMerging property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> timeOfMergingAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the timeOfMerging property

<logical> timeOfMergingIsGettable() Returns true if the timeOfMerging property is gettable for the satellite component class.

<logical> timeOfMergingIsSettable() Specify whether the timeOfMerging property of the satellite component is settable.

<double precision> timeOfMergingRateGet() Returns a zero rate for the timeOfMerging property for the satellite component class.

<void> timeOfMergingSet(<double precision> value) Set the timeOfMerging property of the satellite component.

<type(varying_string)> type() Returns the type name for the standard implementation of the satellite component class.

<double precision, dimension(:), allocatable => classDefault> velocity() Returns the default value for the velocity property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> velocityAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the velocity property

<integer> velocityCount() Compute the count of evolvable quantities in the velocity property of the SatelliteOrbiting component.

<logical> velocityIsGettable() Returns true if the velocity property is gettable for the satellite component class.

<logical> velocityIsSettable() Specify whether the velocity property of the satellite component is settable.

<void> velocityRate(<double precision[:]> value) Cumulate to the rate of the velocity property of the SatelliteOrbiting component.

<double precision, dimension(:), allocatable => classDefault> velocityRateGet() Returns a zero rate for the velocity property for the satellite component class.

<void> velocityScale(<double precision[:]> value) Set the scale of the velocity property of the SatelliteOrbiting component.

<void> velocitySet(<double precision[:]> value) Set the velocity property of the satellite component.

<logical> verySimpleIsActive() Return true if the verySimple implementation of the satellite component is the active choice.

<type(keplerOrbit)> virialOrbit() Get the value of the virialOrbit property of the standard implementation of the satellite component using a deferred function.

<type(varying_string), allocatable, dimension(:) => matches> virialOrbitAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the virialOrbit property

<void> virialOrbitFunction() Set the function to be used for the get method of the virialOrbit property of the SatelliteStandard component.

<logical> virialOrbitIsAttached() Return true if the deferred function used to get the virialOrbit property of the SatelliteStandard component class has been attached.

<logical> virialOrbitIsGettable() Returns true if the virialOrbit property is gettable for the satellite component class.

<logical> virialOrbitIsSettable() Specify whether the virialOrbit property of the satellite component is settable.

<type(keplerOrbit)> virialOrbitRateGet() Returns a zero rate for the virialOrbit property for the satellite component class.

<void> virialOrbitSet(<type(keplerOrbit)> setValue→) Set the value of the virialOrbit property of the standard implementation of the satellite component using a deferred function.

<void> virialOrbitSetFunction() Set the function to be used for the set method of the virialOrbit property of the SatelliteStandard component.

<logical> virialOrbitSetIsAttached() Return true if the deferred function used to set the virialOrbit property of the SatelliteStandard component class has been attached.

<void> virialOrbitSetValue(<type(keplerOrbit)> setValue→) Set the virialOrbit property of an standard implementation of the satellite component class.

<type(keplerOrbit)> virialOrbitValue() Get the virialOrbit property of an standard implementation of the satellite component class.

nodeComponentSatelliteVerySimple

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<double precision> boundMass() Returns the default value for the boundMass property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> boundMassAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the boundMass property

<integer> boundMassCount() Compute the count of evolvable quantities in the boundMass property of the SatelliteOrbiting component.

<type(history)> boundMassHistory() Returns the default value for the boundMassHistory property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> boundMassHistoryAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the boundMassHistory property

<logical> boundMassHistoryIsGettable() Returns true if the boundMassHistory property is gettable for the satellite component class.

<logical> boundMassHistoryIsSettable() Specify whether the boundMassHistory property of the satellite component is settable.

<type(history)> boundMassHistoryRateGet() Returns a zero rate for the boundMassHistory property for the satellite component class.

<void> boundMassHistorySet(<type(history)> value) Set the boundMassHistory property of the satellite component.

<logical> boundMassIsGettable() Returns true if the boundMass property is gettable for the satellite component class.

<logical> boundMassIsSettable() Specify whether the boundMass property of the satellite component is settable.

<void> boundMassRate(<double precision> value) Cumulate to the rate of the boundMass property of the SatelliteOrbiting component.

<double precision> boundMassRateGet() Returns a zero rate for the boundMass property for the satellite component class.

<void> boundMassScale(<double precision> value) Set the scale of the boundMass property of the SatelliteOrbiting component.

<void> boundMassSet(<double precision> value) Set the boundMass property of the satellite component.

<void> builder(<*type(node)> componentDefinition→) Build a verySimple implementation of the satellite component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a verySimple implementation of the satellite component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a verySimple implementation of the satellite component from array.

<void> destroy() Finalize a verySimple implementation of the satellite component.

<void> dumpASCII() Dump the content of a satellite component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a verySimple member of the satellite component.

<logical> isOrphan() Returns the default value for the isOrphan property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> isOrphanAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the isOrphan property

<logical> isOrphanIsGettable() Returns true if the isOrphan property is gettable for the satellite component class.

<logical> isOrphanIsSettable() Specify whether the isOrphan property of the satellite component is settable.

<logical> isOrphanRateGet() Returns a zero rate for the isOrphan property for the satellite component class.

<void> isOrphanSet(<logical> value) Set the isOrphan property of the satellite component.

<double precision> mergeTime() Returns the default value for the mergeTime property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> mergeTimeAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the mergeTime property

<integer> mergeTimeCount() Return a count of the number of scalar properties in the mergeTime property of an verySimple implementation of the satellite component class.

<void> mergeTimeInactive() Indicate that the mergeTime property of an verySimple implementation of the satellite component class is inactive for differential equation solving.

<logical> mergeTimeIsGettable() Returns true if the mergeTime property is gettable for the satellite component class.

<logical> mergeTimeIsSettable() Specify whether the mergeTime property of the satellite component is settable.

<void> mergeTimeRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask) [interruptProcedure]↔) Accumulate to the rate of change of the mergeTime property of an verySimple implementation of the satellite component class.

<double precision> mergeTimeRateGet() Get the rate of change of the mergeTime property of an verySimple implementation of the satellite component class.

<void> mergeTimeScale(<double precision> setValue→) Set the absolute scale of the mergeTime property of an verySimple implementation of the satellite component class.

<void> mergeTimeSet(<double precision> setValue→) Set the mergeTime property of an verySimple implementation of the satellite component class.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a verySimple implementation of the satellite component class.

<integer(kind=kind_int8) => classDefault> nodeIndex() Returns the default value for the nodeIndex property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> nodeIndexAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the nodeIndex property

<type(longIntegerHistory)> nodeIndexHistory() Returns the default value for the nodeIndexHistory property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> nodeIndexHistoryAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the nodeIndexHistory property

<logical> nodeIndexHistoryIsGettable() Returns true if the nodeIndexHistory property is gettable for the satellite component class.

<logical> nodeIndexHistoryIsSettable() Specify whether the nodeIndexHistory property of the satellite component is settable.

<type(longIntegerHistory)> nodeIndexHistoryRateGet() Returns a zero rate for the nodeIndexHistory property for the satellite component class.

<void> nodeIndexHistorySet(<type(longIntegerHistory)> value) Set the nodeIndexHistory property of the satellite component.

<logical> nodeIndexIsGettable() Returns true if the nodeIndex property is gettable for the satellite component class.

<logical> nodeIndexIsSettable() Specify whether the nodeIndex property of the satellite component is settable.

<integer(kind=kind_int8) => classDefault> nodeIndexRateGet() Returns a zero rate for the nodeIndex property for the satellite component class.

<logical> nullIsActive() Return true if the null implementation of the satellite component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<logical> orbitingIsActive() Return true if the orbiting implementation of the satellite component is the active choice.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a satellite component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a verySimple implementation of the satellite component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a verySimple implementation of the satellite component.

<double precision, dimension(:), allocatable => classDefault> position() Returns the default value for the position property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> positionAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the position property

<integer> positionCount() Compute the count of evolvable quantities in the position property of the SatelliteOrbiting component.

<logical> positionIsGettable() Returns true if the position property is gettable for the satellite component class.

<logical> positionIsSettable() Specify whether the position property of the satellite component is settable.

<void> positionRate(<double precision[:]> value) Cumulate to the rate of the position property of the SatelliteOrbiting component.

<double precision, dimension(:), allocatable => classDefault> positionRateGet() Returns a zero rate for the position property for the satellite component class.

<void> positionScale(<double precision[:]> value) Set the scale of the position property of the SatelliteOrbiting component.

<void> positionSet(<double precision[:]> value) Set the position property of the satellite component.

<void> postOutput(<double precision> time→) Perform post-output processing of a satellite component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<logical> presetIsActive() Return true if the preset implementation of the satellite component is the active choice.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a verySimple implementation of the satellite component.

<void> serializeASCII() Serialize the contents of a verySimple implementation of the satellite component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a verySimple implementation of the satellite component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a verySimple implementation of the satellite component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a verySimple implementation of the satellite component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a verySimple implementation of the satellite component to XML.

<integer(c_size_t)> sizeOf() Return the size in bytes of a verySimple implementation of the satellite component.

<logical> standardIsActive() Return true if the standard implementation of the satellite component is the active choice.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<double precision> tidalHeatingNormalized() Returns the default value for the tidalHeatingNormalized property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> tidalHeatingNormalizedAttributeMatch(<log [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the tidalHeatingNormalized property

<integer> tidalHeatingNormalizedCount() Compute the count of evolvable quantities in the tidalHeatingNormalized property of the SatelliteOrbiting component.

<logical> tidalHeatingNormalizedIsGettable() Returns true if the tidalHeatingNormalized property is gettable for the satellite component class.

<logical> tidalHeatingNormalizedIsSettable() Specify whether the tidalHeatingNormalized property of the satellite component is settable.

<void> tidalHeatingNormalizedRate(<double precision> value) Cumulate to the rate of the tidalHeatingNormalize property of the SatelliteOrbiting component.

<double precision> tidalHeatingNormalizedRateGet() Returns a zero rate for the tidalHeatingNormalized property for the satellite component class.

<void> tidalHeatingNormalizedScale(<double precision> value) Set the scale of the tidalHeatingNormalized property of the SatelliteOrbiting component.

<void> tidalHeatingNormalizedSet(<double precision> value) Set the tidalHeatingNormalized property of the satellite component.

<type(tensorRank2Dimension3Symmetric)> tidalTensorPathIntegrated() Returns the default value for the tidalTensorPathIntegrated property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> tidalTensorPathIntegratedAttributeMatch(< [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the tidalTensorPathIntegrated property

<integer> tidalTensorPathIntegratedCount() Compute the count of evolvable quantities in the tidalTensorPathIntegrated property of the SatelliteOrbiting component.

<logical> tidalTensorPathIntegratedIsGettable() Returns true if the tidalTensorPathIntegrated property is gettable for the satellite component class.

<logical> tidalTensorPathIntegratedIsSettable() Specify whether the tidalTensorPathIntegrated property of the satellite component is settable.

<void> tidalTensorPathIntegratedRate(<type(tensorRank2Dimension3Symmetric)> value) Cumulate to the rate of the tidalTensorPathIntegrated property of the SatelliteOrbiting component.

<type(tensorRank2Dimension3Symmetric)> tidalTensorPathIntegratedRateGet() Returns a zero rate for the tidalTensorPathIntegrated property for the satellite component class.

<void> tidalTensorPathIntegratedScale(<type(tensorRank2Dimension3Symmetric)> value) Set the scale of the tidalTensorPathIntegrated property of the SatelliteOrbiting component.

<void> tidalTensorPathIntegratedSet(<type(tensorRank2Dimension3Symmetric)> value) Set the tidalTensorPathIntegrated property of the satellite component.

<double precision> timeOfMerging() Returns the default value for the timeOfMerging property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> timeOfMergingAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the timeOfMerging property

<logical> timeOfMergingIsGettable() Returns true if the timeOfMerging property is gettable for the satellite component class.

<logical> timeOfMergingIsSettable() Specify whether the timeOfMerging property of the satellite component is settable.

<double precision> timeOfMergingRateGet() Returns a zero rate for the timeOfMerging property for the satellite component class.

<void> timeOfMergingSet(<double precision> value) Set the timeOfMerging property of the satellite component.

<type(varying_string)> type() Returns the type name for the verySimple implementation of the satellite component class.

<double precision, dimension(:), allocatable => classDefault> velocity() Returns the default value for the velocity property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> velocityAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the velocity property

<integer> velocityCount() Compute the count of evolvable quantities in the velocity property of the SatelliteOrbiting component.

<logical> velocityIsGettable() Returns true if the velocity property is gettable for the satellite component class.

<logical> velocityIsSettable() Specify whether the velocity property of the satellite component is settable.

<void> velocityRate(<double precision[:]> value) Cumulate to the rate of the velocity property of the SatelliteOrbiting component.

<double precision, dimension(:), allocatable => classDefault> velocityRateGet() Returns a zero rate for the velocity property for the satellite component class.

<void> velocityScale(<double precision[:]> value) Set the scale of the velocity property of the SatelliteOrbiting component.

<void> velocitySet(<double precision[:]> value) Set the velocity property of the satellite component.

<logical> verySimpleIsActive() Return true if the verySimple implementation of the satellite component is the active choice.

<type(keplerOrbit)> virialOrbit() Returns the default value for the virialOrbit property for the satellite component class.

<type(varying_string), allocatable, dimension(:) => matches> virialOrbitAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the satellite class that have the desired attributes for the virialOrbit property

<logical> virialOrbitIsGettable() Returns true if the virialOrbit property is gettable for the satellite component class.

<logical> virialOrbitIsSettable() Specify whether the virialOrbit property of the satellite component is settable.

<type(keplerOrbit)> virialOrbitRateGet() Returns a zero rate for the virialOrbit property for the satellite component class.

<void> virialOrbitSet(<type(keplerOrbit)> value) Set the virialOrbit property of the satellite component.

nodeComponentSpheroid

<type(abundances)> abundancesGas() Returns the default value for the abundancesGas property for the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> abundancesGasAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spheroid class that have the desired attributes for the abundancesGas property

<integer> abundancesGasCount() Compute the count of evolvable quantities in the abundancesGas property of the SpheroidStandard component.

<logical> abundancesGasIsGettable() Returns true if the abundancesGas property is gettable for the spheroid component class.

<logical> abundancesGasIsSettable() Specify whether the abundancesGas property of the spheroid component is settable.

<void> abundancesGasRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptT [interruptProcedure]↔) Accept a rate set for the abundancesGas property of the spheroid component class. Trigger an interrupt to create the component.

<type(abundances)> abundancesGasRateGet() Returns a zero rate for the abundancesGas property for the spheroid component class.

<void> abundancesGasScale(<type(abundances)> value) Set the scale of the abundancesGas property of the SpheroidStandard component.

<void> abundancesGasSet(<type(abundances)> value) Set the abundancesGas property of the spheroid component.

<type(abundances)> abundancesStellar() Returns the default value for the abundancesStellar property for the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> abundancesStellarAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spheroid class that have the desired attributes for the abundancesStellar property

<integer> abundancesStellarCount() Compute the count of evolvable quantities in the abundancesStellar property of the SpheroidStandard component.

<logical> abundancesStellarIsGettable() Returns true if the abundancesStellar property is gettable for the spheroid component class.

<logical> abundancesStellarIsSettable() Specify whether the abundancesStellar property of the spheroid component is settable.

<void> abundancesStellarRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interr [interruptProcedure]↔) Accept a rate set for the abundancesStellar property of the spheroid component class. Trigger an interrupt to create the component.

<type(abundances)> abundancesStellarRateGet() Returns a zero rate for the abundancesStellar property for the spheroid component class.

<void> abundancesStellarScale(<type(abundances)> value) Set the scale of the abundancesStellar property of the SpheroidStandard component.

<void> abundancesStellarSet(<type(abundances)> value) Set the abundancesStellar property of the spheroid component.

<double precision> angularMomentum() Returns the default value for the angularMomentum property for the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> angularMomentumAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spheroid class that have the desired attributes for the angularMomentum property

<integer> angularMomentumCount() Compute the count of evolvable quantities in the angularMomentum property of the SpheroidStandard component.

<logical> angularMomentumIsGettable() Returns true if the angularMomentum property is gettable for the spheroid component class.

<logical> angularMomentumIsSettable() Specify whether the angularMomentum property of the spheroid component is settable.

<void> angularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interrup [interruptProcedure]↔) Accept a rate set for the angularMomentum property of the spheroid component class. Trigger an interrupt to create the component.

<double precision> angularMomentumRateGet() Returns a zero rate for the angularMomentum property for the spheroid component class.

<void> angularMomentumScale(<double precision> value) Set the scale of the angularMomentum property of the SpheroidStandard component.

<void> angularMomentumSet(<double precision> value) Set the angularMomentum property of the spheroid component.

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a generic spheroid component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Read properties from raw file.

<void> deserializeValues(<double(:)> array→, <integer> propertyType→) Deserialize the evolvable quantities from an array.

<void> destroy() Finalize a generic spheroid component.

<void> dumpASCII() Dump the content of a spheroid component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<double precision> energyGasInput() Returns the default value for the energyGasInput property for the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> energyGasInputAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spheroid class that have the desired attributes for the energyGasInput property

<integer> energyGasInputCount() Compute the count of evolvable quantities in the energyGasInput property of the SpheroidStandard component.

<logical> energyGasInputIsGettable() Returns true if the energyGasInput property is gettable for the spheroid component class.

<logical> energyGasInputIsSettable() Specify whether the energyGasInput property of the spheroid component is settable.

<void> energyGasInputRate(<double precision> value) Cumulate to the rate of the energyGasInput property of the SpheroidStandard component.

<double precision> energyGasInputRateGet() Returns a zero rate for the energyGasInput property for the spheroid component class.

<double precision> halfMassRadius() Returns the default value for the halfMassRadius property for the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> halfMassRadiusAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spheroid class that have the desired attributes for the halfMassRadius property

<logical> halfMassRadiusIsGettable() Returns true if the halfMassRadius property is gettable for the spheroid component class.

<logical> halfMassRadiusIsSettable() Specify whether the halfMassRadius property of the spheroid component is settable.

<double precision> halfMassRadiusRateGet() Returns a zero rate for the halfMassRadius property for the spheroid component class.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a generic spheroid component.

<logical> isInitialized() Returns the default value for the isInitialized property for the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> isInitializedAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spheroid class that have the desired attributes for the isInitialized property

<logical> isInitializedIsGettable() Returns true if the isInitialized property is gettable for the spheroid component class.

<logical> isInitializedIsSettable() Specify whether the isInitialized property of the spheroid component is settable.

<logical> isInitializedRateGet() Returns a zero rate for the isInitialized property for the spheroid component class.

<void> isInitializedSet(<logical> value) Set the isInitialized property of the spheroid component.

<type(stellarLuminosities)> luminositiesStellar() Returns the default value for the luminositiesStellar property for the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> luminositiesStellarAttributeMatch(<logica
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the spheroid class that have the desired attributes for the luminositiesStellar property

<integer> luminositiesStellarCount() Compute the count of evolvable quantities in the luminositiesStellar property of the SpheroidStandard component.

<logical> luminositiesStellarIsGettable() Returns true if the luminositiesStellar property is gettable for the spheroid component class.

<logical> luminositiesStellarIsSettable() Specify whether the luminositiesStellar property of the spheroid component is settable.

<void> luminositiesStellarRate(<type(stellarLuminosities)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accept a rate set for the luminositiesStellar property of the spheroid component class. Trigger an interrupt to create the component.

<type(stellarLuminosities)> luminositiesStellarRateGet() Returns a zero rate for the luminositiesStellar property for the spheroid component class.

<void> luminositiesStellarScale(<type(stellarLuminosities)> value) Set the scale of the luminositiesStellar property of the SpheroidStandard component.

<void> luminositiesStellarSet(<type(stellarLuminosities)> value) Set the luminositiesStellar property of the spheroid component.

<double precision> massGas() Returns the default value for the massGas property for the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> massGasAttributeMatch(<logical>
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the spheroid class that have the desired attributes for the massGas property

<integer> massGasCount() Compute the count of evolvable quantities in the massGas property of the SpheroidStandard component.

<logical> massGasIsGettable() Returns true if the massGas property is gettable for the spheroid component class.

<logical> massGasIsSettable() Specify whether the massGas property of the spheroid component is settable.

<void> massGasRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)>
[interruptProcedure]↔) Accept a rate set for the massGas property of the spheroid component class. Trigger an interrupt to create the component.

<double precision> massGasRateGet() Returns a zero rate for the massGas property for the spheroid component class.

<void> massGasScale(<double precision> value) Set the scale of the massGas property of the SpheroidStandard component.

<void> massGasSet(<double precision> value) Set the massGas property of the spheroid component.

<double precision> massGasSink() Returns the default value for the massGasSink property for the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> massGasSinkAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spheroid class that have the desired attributes for the massGasSink property

<integer> massGasSinkCount() Compute the count of evolvable quantities in the massGasSink property of the SpheroidStandard component.

<logical> massGasSinkIsGettable() Returns true if the massGasSink property is gettable for the spheroid component class.

<logical> massGasSinkIsSettable() Specify whether the massGasSink property of the spheroid component is settable.

<void> massGasSinkRate(<double precision> value) Cumulate to the rate of the massGasSink property of the SpheroidStandard component.

<double precision> massGasSinkRateGet() Returns a zero rate for the massGasSink property for the spheroid component class.

<double precision> massStellar() Returns the default value for the massStellar property for the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> massStellarAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spheroid class that have the desired attributes for the massStellar property

<integer> massStellarCount() Compute the count of evolvable quantities in the massStellar property of the SpheroidStandard component.

<double precision> massStellarFormed() Returns the default value for the massStellarFormed property for the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> massStellarFormedAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spheroid class that have the desired attributes for the massStellarFormed property

<integer> massStellarFormedCount() Compute the count of evolvable quantities in the massStellarFormed property of the SpheroidStandard component.

<logical> massStellarFormedIsGettable() Returns true if the massStellarFormed property is gettable for the spheroid component class.

<logical> massStellarFormedIsSettable() Specify whether the massStellarFormed property of the spheroid component is settable.

<void> massStellarFormedRate(<double precision> value) Cumulate to the rate of the massStellarFormed property of the SpheroidStandard component.

<double precision> massStellarFormedRateGet() Returns a zero rate for the massStellarFormed property for the spheroid component class.

<void> massStellarFormedScale(<double precision> value) Set the scale of the massStellarFormed property of the SpheroidStandard component.

<void> massStellarFormedSet(<double precision> value) Set the massStellarFormed property of the spheroid component.

<logical> massStellarIsGettable() Returns true if the massStellar property is gettable for the spheroid component class.

<logical> massStellarIsSettable() Specify whether the massStellar property of the spheroid component is settable.

<void> massStellarRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTas [interruptProcedure]↔) Accept a rate set for the massStellar property of the spheroid component class. Trigger an interrupt to create the component.

<double precision> massStellarRateGet() Returns a zero rate for the massStellar property for the spheroid component class.

<void> massStellarScale(<double precision> value) Set the scale of the massStellar property of the SpheroidStandard component.

<void> massStellarSet(<double precision> value) Set the massStellar property of the spheroid component.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a nodeComponent object.

<logical> nullIsActive() Return true if the null implementation of the spheroid component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a spheroid component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a generic spheroid component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Establish the names of properties to output for a generic spheroid component.

<void> postOutput(<double precision> time→) Perform post-output processing of a spheroid component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<double precision> radius() Returns the default value for the radius property for the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> radiusAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spheroid class that have the desired attributes for the radius property

<logical> radiusIsGettable() Returns true if the radius property is gettable for the spheroid component class.

<logical> radiusIsSettable() Specify whether the radius property of the spheroid component is settable.

<double precision> radiusRateGet() Returns a zero rate for the radius property for the spheroid component class.

<void> radiusSet(<double precision> value) Set the radius property of the spheroid component.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets() Set offsets into serialization arrays.

<void> serializeASCII() Serialize the content of a spheroid component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the number of evolvable quantities to be evolved.

<void> serializeRaw(<integer> fileHandle→) Generate a binary dump of all properties.

<void> serializeValues(<double(:)> array←, <integer> propertyType→) Serialize the evolvable quantities to an array.

<void> serializeXML() Generate an XML dump of all properties.

<integer(c_size_t)> sizeOf() Return the size in bytes of a nodeComponentSpheroid component.

<logical> standardIsActive() Return true if the standard implementation of the spheroid component is the active choice.

<type(history)> starFormationHistory() Returns the default value for the starFormationHistory property for the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> starFormationHistoryAttributeMatch(<logic
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spheroid class that have the desired attributes for the starFormationHistory property

<integer> starFormationHistoryCount() Compute the count of evolvable quantities in the starFormationHistory property of the SpheroidStandard component.

<logical> starFormationHistoryIsGettable() Returns true if the starFormationHistory property is gettable for the spheroid component class.

<logical> starFormationHistoryIsSettable() Specify whether the starFormationHistory property of the spheroid component is settable.

<void> starFormationHistoryRate(<type(history)> setValue→, <logical> [interrupt]↔, <*procedure(interr [interruptProcedure]↔) Accept a rate set for the starFormationHistory property of the spheroid component class. Trigger an interrupt to create the component.

<type(history)> starFormationHistoryRateGet() Returns a zero rate for the starFormationHistory property for the spheroid component class.

<void> starFormationHistoryScale(<type(history)> value) Set the scale of the starFormationHistory property of the SpheroidStandard component.

<void> starFormationHistorySet(<type(history)> value) Set the starFormationHistory property of the spheroid component.

<double precision> starFormationRate() Returns the default value for the starFormationRate property for the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> starFormationRateAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spheroid class that have the desired attributes for the starFormationRate property

<logical> starFormationRateIsGettable() Returns true if the starFormationRate property is gettable for the spheroid component class.

<logical> starFormationRateIsSettable() Specify whether the starFormationRate property of the spheroid component is settable.

<double precision> starFormationRateRateGet() Returns a zero rate for the starFormationRate property for the spheroid component class.

<type(history)> stellarPropertiesHistory() Returns the default value for the stellarPropertiesHistory property for the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> stellarPropertiesHistoryAttributeMatch(<l [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spheroid class that have the desired attributes for the stellarPropertiesHistory property

<integer> stellarPropertiesHistoryCount() Compute the count of evolvable quantities in the stellarPropertiesHist property of the SpheroidStandard component.

<logical> stellarPropertiesHistoryIsGettable() Returns true if the stellarPropertiesHistory property is gettable for the spheroid component class.

<logical> stellarPropertiesHistoryIsSettable() Specify whether the stellarPropertiesHistory property of the spheroid component is settable.

<void> stellarPropertiesHistoryRate(<type(history)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accept a rate set for the stellarPropertiesHistory property of the spheroid component class. Trigger an interrupt to create the component.

<type(history)> stellarPropertiesHistoryRateGet() Returns a zero rate for the stellarPropertiesHistory property for the spheroid component class.

<void> stellarPropertiesHistoryScale(<type(history)> value) Set the scale of the stellarPropertiesHistory property of the SpheroidStandard component.

<void> stellarPropertiesHistorySet(<type(history)> value) Set the stellarPropertiesHistory property of the spheroid component.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<type(varying_string)> type() Returns the type name for the spheroid component class.

<double precision> velocity() Returns the default value for the velocity property for the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> velocityAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spheroid class that have the desired attributes for the velocity property

<logical> velocityIsGettable() Returns true if the velocity property is gettable for the spheroid component class.

<logical> velocityIsSettable() Specify whether the velocity property of the spheroid component is settable.

<double precision> velocityRateGet() Returns a zero rate for the velocity property for the spheroid component class.

<void> velocitySet(<double precision> value) Set the velocity property of the spheroid component.

<logical> verySimpleIsActive() Return true if the verySimple implementation of the spheroid component is the active choice.

nodeComponentSpheroidNull

<type(abundances)> abundancesGas() Returns the default value for the abundancesGas property for the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> abundancesGasAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spheroid class that have the desired attributes for the abundancesGas property

<integer> abundancesGasCount() Compute the count of evolvable quantities in the abundancesGas property of the SpheroidStandard component.

<logical> abundancesGasIsGettable() Returns true if the abundancesGas property is gettable for the spheroid component class.

<logical> abundancesGasIsSettable() Specify whether the abundancesGas property of the spheroid component is settable.

<void> abundancesGasRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptT
[interruptProcedure]↔) Accept a rate set for the abundancesGas property of the spheroid component class. Trigger an interrupt to create the component.

<type(abundances)> abundancesGasRateGet() Returns a zero rate for the abundancesGas property for the spheroid component class.

<void> abundancesGasScale(<type(abundances)> value) Set the scale of the abundancesGas property of the SpheroidStandard component.

<void> abundancesGasSet(<type(abundances)> value) Set the abundancesGas property of the spheroid component.

<type(abundances)> abundancesStellar() Returns the default value for the abundancesStellar property for the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> abundancesStellarAttributeMatch(<logical>
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the spheroid class that have the desired attributes for the abundancesStellar property

<integer> abundancesStellarCount() Compute the count of evolvable quantities in the abundancesStellar property of the SpheroidStandard component.

<logical> abundancesStellarIsGettable() Returns true if the abundancesStellar property is gettable for the spheroid component class.

<logical> abundancesStellarIsSettable() Specify whether the abundancesStellar property of the spheroid component is settable.

<void> abundancesStellarRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interr
[interruptProcedure]↔) Accept a rate set for the abundancesStellar property of the spheroid component class. Trigger an interrupt to create the component.

<type(abundances)> abundancesStellarRateGet() Returns a zero rate for the abundancesStellar property for the spheroid component class.

<void> abundancesStellarScale(<type(abundances)> value) Set the scale of the abundancesStellar property of the SpheroidStandard component.

<void> abundancesStellarSet(<type(abundances)> value) Set the abundancesStellar property of the spheroid component.

<double precision> angularMomentum() Returns the default value for the angularMomentum property for the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> angularMomentumAttributeMatch(<logical>
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the spheroid class that have the desired attributes for the angularMomentum property

<integer> angularMomentumCount() Compute the count of evolvable quantities in the angularMomentum property of the SpheroidStandard component.

<logical> angularMomentumIsGettable() Returns true if the angularMomentum property is gettable for the spheroid component class.

<logical> angularMomentumIsSettable() Specify whether the angularMomentum property of the spheroid component is settable.

<void> angularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interrup [interruptProcedure]↔) Accept a rate set for the angularMomentum property of the spheroid component class. Trigger an interrupt to create the component.

<double precision> angularMomentumRateGet() Returns a zero rate for the angularMomentum property for the spheroid component class.

<void> angularMomentumScale(<double precision> value) Set the scale of the angularMomentum property of the SpheroidStandard component.

<void> angularMomentumSet(<double precision> value) Set the angularMomentum property of the spheroid component.

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a null implementation of the spheroid component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a null implementation of the spheroid component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a null implementation of the spheroid component from array.

<void> destroy() Finalize a null implementation of the spheroid component.

<void> dumpASCII() Dump the content of a spheroid component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<double precision> energyGasInput() Returns the default value for the energyGasInput property for the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> energyGasInputAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spheroid class that have the desired attributes for the energyGasInput property

<integer> energyGasInputCount() Compute the count of evolvable quantities in the energyGasInput property of the SpheroidStandard component.

<logical> energyGasInputIsGettable() Returns true if the energyGasInput property is gettable for the spheroid component class.

<logical> energyGasInputIsSettable() Specify whether the energyGasInput property of the spheroid component is settable.

<void> energyGasInputRate(<double precision> value) Cumulate to the rate of the energyGasInput property of the SpheroidStandard component.

<double precision> energyGasInputRateGet() Returns a zero rate for the energyGasInput property for the spheroid component class.

<double precision> halfMassRadius() Returns the default value for the halfMassRadius property for the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> halfMassRadiusAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spheroid class that have the desired attributes for the halfMassRadius property

<logical> halfMassRadiusIsGettable() Returns true if the halfMassRadius property is gettable for the spheroid component class.

<logical> halfMassRadiusIsSettable() Specify whether the halfMassRadius property of the spheroid component is settable.

<double precision> halfMassRadiusRateGet() Returns a zero rate for the halfMassRadius property for the spheroid component class.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a null member of the spheroid component.

<logical> isInitialized() Returns the default value for the isInitialized property for the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> isInitializedAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spheroid class that have the desired attributes for the isInitialized property

<logical> isInitializedIsGettable() Returns true if the isInitialized property is gettable for the spheroid component class.

<logical> isInitializedIsSettable() Specify whether the isInitialized property of the spheroid component is settable.

<logical> isInitializedRateGet() Returns a zero rate for the isInitialized property for the spheroid component class.

<void> isInitializedSet(<logical> value) Set the isInitialized property of the spheroid component.

<type(stellarLuminosities)> luminositiesStellar() Returns the default value for the luminositiesStellar property for the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> luminositiesStellarAttributeMatch(<logica [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spheroid class that have the desired attributes for the luminositiesStellar property

<integer> luminositiesStellarCount() Compute the count of evolvable quantities in the luminositiesStellar property of the SpheroidStandard component.

<logical> luminositiesStellarIsGettable() Returns true if the luminositiesStellar property is gettable for the spheroid component class.

<logical> luminositiesStellarIsSettable() Specify whether the luminositiesStellar property of the spheroid component is settable.

<void> luminositiesStellarRate(<type(stellarLuminosities)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accept a rate set for the luminositiesStellar property of the spheroid component class. Trigger an interrupt to create the component.

<type(stellarLuminosities)> luminositiesStellarRateGet() Returns a zero rate for the luminositiesStellar property for the spheroid component class.

<void> luminositiesStellarScale(<type(stellarLuminosities)> value) Set the scale of the luminositiesStellar property of the SpheroidStandard component.

<void> luminositiesStellarSet(<type(stellarLuminosities)> value) Set the luminositiesStellar property of the spheroid component.

<double precision> massGas() Returns the default value for the massGas property for the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> massGasAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spheroid class that have the desired attributes for the massGas property

<integer> massGasCount() Compute the count of evolvable quantities in the massGas property of the SpheroidStandard component.

<logical> massGasIsGettable() Returns true if the massGas property is gettable for the spheroid component class.

<logical> massGasIsSettable() Specify whether the massGas property of the spheroid component is settable.

<void> massGasRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accept a rate set for the massGas property of the spheroid component class. Trigger an interrupt to create the component.

<double precision> massGasRateGet() Returns a zero rate for the massGas property for the spheroid component class.

<void> massGasScale(<double precision> value) Set the scale of the massGas property of the SpheroidStandard component.

<void> massGasSet(<double precision> value) Set the massGas property of the spheroid component.

<double precision> massGasSink() Returns the default value for the massGasSink property for the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> massGasSinkAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spheroid class that have the desired attributes for the massGasSink property

<integer> massGasSinkCount() Compute the count of evolvable quantities in the massGasSink property of the SpheroidStandard component.

<logical> massGasSinkIsGettable() Returns true if the massGasSink property is gettable for the spheroid component class.

<logical> massGasSinkIsSettable() Specify whether the massGasSink property of the spheroid component is settable.

<void> massGasSinkRate(<double precision> value) Cumulate to the rate of the massGasSink property of the SpheroidStandard component.

<double precision> massGasSinkRateGet() Returns a zero rate for the massGasSink property for the spheroid component class.

<double precision> massStellar() Returns the default value for the massStellar property for the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> massStellarAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spheroid class that have the desired attributes for the massStellar property

<integer> massStellarCount() Compute the count of evolvable quantities in the massStellar property of the SpheroidStandard component.

<double precision> massStellarFormed() Returns the default value for the massStellarFormed property for the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> massStellarFormedAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spheroid class that have the desired attributes for the massStellarFormed property

<integer> massStellarFormedCount() Compute the count of evolvable quantities in the massStellarFormed property of the SpheroidStandard component.

<logical> massStellarFormedIsGettable() Returns true if the massStellarFormed property is gettable for the spheroid component class.

<logical> massStellarFormedIsSettable() Specify whether the massStellarFormed property of the spheroid component is settable.

<void> massStellarFormedRate(<double precision> value) Cumulate to the rate of the massStellarFormed property of the SpheroidStandard component.

<double precision> massStellarFormedRateGet() Returns a zero rate for the massStellarFormed property for the spheroid component class.

<void> massStellarFormedScale(<double precision> value) Set the scale of the massStellarFormed property of the SpheroidStandard component.

<void> massStellarFormedSet(<double precision> value) Set the massStellarFormed property of the spheroid component.

<logical> massStellarIsGettable() Returns true if the massStellar property is gettable for the spheroid component class.

<logical> massStellarIsSettable() Specify whether the massStellar property of the spheroid component is settable.

<void> massStellarRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTas [interruptProcedure]↔) Accept a rate set for the massStellar property of the spheroid component class. Trigger an interrupt to create the component.

<double precision> massStellarRateGet() Returns a zero rate for the massStellar property for the spheroid component class.

<void> massStellarScale(<double precision> value) Set the scale of the massStellar property of the SpheroidStandard component.

<void> massStellarSet(<double precision> value) Set the massStellar property of the spheroid component.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a null implementation of the spheroid component class.

<logical> nullIsActive() Return true if the null implementation of the spheroid component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_- int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a spheroid component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a null implementation of the spheroid component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a null implementation of the spheroid component.

<void> postOutput(<double precision> time→) Perform post-output processing of a spheroid component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<double precision> radius() Returns the default value for the radius property for the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> radiusAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spheroid class that have the desired attributes for the radius property

<logical> radiusIsGettable() Returns true if the radius property is gettable for the spheroid component class.

<logical> radiusIsSettable() Specify whether the radius property of the spheroid component is settable.

<double precision> radiusRateGet() Returns a zero rate for the radius property for the spheroid component class.

<void> radiusSet(<double precision> value) Set the radius property of the spheroid component.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a null implementation of the spheroid component.

<void> serializeASCII() Serialize the contents of a null implementation of the spheroid component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a null implementation of the spheroid component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a null implementation of the spheroid component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a null implementation of the spheroid component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a null implementation of the spheroid component to XML.

<integer(c_size_t)> sizeOf() Return the size in bytes of a null implementation of the spheroid component.

<logical> standardIsActive() Return true if the standard implementation of the spheroid component is the active choice.

<type(history)> starFormationHistory() Returns the default value for the starFormationHistory property for the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> starFormationHistoryAttributeMatch(<logic [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spheroid class that have the desired attributes for the starFormationHistory property

<integer> starFormationHistoryCount() Compute the count of evolvable quantities in the starFormationHistory property of the SpheroidStandard component.

<logical> starFormationHistoryIsGettable() Returns true if the starFormationHistory property is gettable for the spheroid component class.

<logical> starFormationHistoryIsSettable() Specify whether the starFormationHistory property of the spheroid component is settable.

<void> starFormationHistoryRate(<type(history)> setValue→, <logical> [interrupt]↔, <*procedure(interr [interruptProcedure]↔) Accept a rate set for the starFormationHistory property of the spheroid component class. Trigger an interrupt to create the component.

<type(history)> starFormationHistoryRateGet() Returns a zero rate for the starFormationHistory property for the spheroid component class.

<void> starFormationHistoryScale(<type(history)> value) Set the scale of the starFormationHistory property of the SpheroidStandard component.

<void> starFormationHistorySet(<type(history)> value) Set the starFormationHistory property of the spheroid component.

<double precision> starFormationRate() Returns the default value for the starFormationRate property for the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> starFormationRateAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spheroid class that have the desired attributes for the starFormationRate property

<logical> starFormationRateIsGettable() Returns true if the starFormationRate property is gettable for the spheroid component class.

<logical> starFormationRateIsSettable() Specify whether the starFormationRate property of the spheroid component is settable.

<double precision> starFormationRateRateGet() Returns a zero rate for the starFormationRate property for the spheroid component class.

<type(history)> stellarPropertiesHistory() Returns the default value for the stellarPropertiesHistory property for the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> stellarPropertiesHistoryAttributeMatch(<l
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the spheroid class that have the desired attributes for the stellarPropertiesHistory property

<integer> stellarPropertiesHistoryCount() Compute the count of evolvable quantities in the stellarPropertiesHist
property of the SpheroidStandard component.

<logical> stellarPropertiesHistoryIsGettable() Returns true if the stellarPropertiesHistory property is gettable for the spheroid component class.

<logical> stellarPropertiesHistoryIsSettable() Specify whether the stellarPropertiesHistory property of the spheroid component is settable.

<void> stellarPropertiesHistoryRate(<type(history)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accept a rate set for the stellarPropertiesHistory property of the spheroid component class. Trigger an interrupt to create the component.

<type(history)> stellarPropertiesHistoryRateGet() Returns a zero rate for the stellarPropertiesHistory property for the spheroid component class.

<void> stellarPropertiesHistoryScale(<type(history)> value) Set the scale of the stellarPropertiesHistory property of the SpheroidStandard component.

<void> stellarPropertiesHistorySet(<type(history)> value) Set the stellarPropertiesHistory property of the spheroid component.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<type(varying_string)> type() Returns the type name for the null implementation of the spheroid component class.

<double precision> velocity() Returns the default value for the velocity property for the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> velocityAttributeMatch(<logical>
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the spheroid class that have the desired attributes for the velocity property

<logical> velocityIsGettable() Returns true if the velocity property is gettable for the spheroid component class.

<logical> velocityIsSettable() Specify whether the velocity property of the spheroid component is settable.

<double precision> velocityRateGet() Returns a zero rate for the velocity property for the spheroid component class.

<void> velocitySet(<double precision> value) Set the velocity property of the spheroid component.

<logical> verySimpleIsActive() Return true if the verySimple implementation of the spheroid component is the active choice.

nodeComponentSpheroidStandard

<type(abundances)> abundancesGas() Get the abundancesGas property of an standard implementation of the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> abundancesGasAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spheroid class that have the desired attributes for the abundancesGas property

<integer> abundancesGasCount() Return a count of the number of scalar properties in the abundancesGas property of an standard implementation of the spheroid component class.

<void> abundancesGasInactive() Indicate that the abundancesGas property of an standard implementation of the spheroid component class is inactive for differential equation solving.

<logical> abundancesGasIsGettable() Returns true if the abundancesGas property is gettable for the spheroid component class.

<logical> abundancesGasIsSettable() Specify whether the abundancesGas property of the spheroid component is settable.

<void> abundancesGasRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptT [interruptProcedure]↔) Accumulate to the rate of change of the abundancesGas property of an standard implementation of the spheroid component class.

<type(abundances)> abundancesGasRateGet() Get the rate of change of the abundancesGas property of an standard implementation of the spheroid component class.

<void> abundancesGasScale(<type(abundances)> setValue→) Set the absolute scale of the abundancesGas property of an standard implementation of the spheroid component class.

<void> abundancesGasSet(<type(abundances)> setValue→) Set the abundancesGas property of an standard implementation of the spheroid component class.

<type(abundances)> abundancesStellar() Get the abundancesStellar property of an standard implementation of the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> abundancesStellarAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spheroid class that have the desired attributes for the abundancesStellar property

<integer> abundancesStellarCount() Return a count of the number of scalar properties in the abundancesStellar property of an standard implementation of the spheroid component class.

<void> abundancesStellarInactive() Indicate that the abundancesStellar property of an standard implementation of the spheroid component class is inactive for differential equation solving.

<logical> abundancesStellarIsGettable() Returns true if the abundancesStellar property is gettable for the spheroid component class.

<logical> abundancesStellarIsSettable() Specify whether the abundancesStellar property of the spheroid component is settable.

<void> abundancesStellarRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interr
[interruptProcedure]↔) Accumulate to the rate of change of the abundancesStellar property of an standard implementation of the spheroid component class.

<type(abundances)> abundancesStellarRateGet() Get the rate of change of the abundancesStellar property of an standard implementation of the spheroid component class.

<void> abundancesStellarScale(<type(abundances)> setValue→) Set the absolute scale of the abundancesStellar property of an standard implementation of the spheroid component class.

<void> abundancesStellarSet(<type(abundances)> setValue→) Set the abundancesStellar property of an standard implementation of the spheroid component class.

<double precision> angularMomentum() Get the angularMomentum property of an standard implementation of the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> angularMomentumAttributeMatch(<logical>
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the spheroid class that have the desired attributes for the angularMomentum property

<integer> angularMomentumCount() Return a count of the number of scalar properties in the angularMomentum property of an standard implementation of the spheroid component class.

<void> angularMomentumInactive() Indicate that the angularMomentum property of an standard implementation of the spheroid component class is inactive for differential equation solving.

<logical> angularMomentumIsGettable() Returns true if the angularMomentum property is gettable for the spheroid component class.

<logical> angularMomentumIsSettable() Specify whether the angularMomentum property of the spheroid component is settable.

<void> angularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interrup
[interruptProcedure]↔) Accumulate to the rate of change of the angularMomentum property of an standard implementation of the spheroid component class.

<double precision> angularMomentumRateGet() Get the rate of change of the angularMomentum property of an standard implementation of the spheroid component class.

<void> angularMomentumScale(<double precision> setValue→) Set the absolute scale of the angularMomentum property of an standard implementation of the spheroid component class.

<void> angularMomentumSet(<double precision> setValue→) Set the angularMomentum property of an standard implementation of the spheroid component class.

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a standard implementation of the spheroid component from a supplied XML definition.

<void> createFunctionSet() Set the create function for the standard implementation of the spheroid component class.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a standard implementation of the spheroid component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a standard implementation of the spheroid component from array.

<void> destroy() Finalize a standard implementation of the spheroid component.

<void> dumpASCII() Dump the content of a spheroid component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<double precision> energyGasInput() Returns the default value for the energyGasInput property for the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> energyGasInputAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spheroid class that have the desired attributes for the energyGasInput property

<integer> energyGasInputCount() Compute the count of evolvable quantities in the energyGasInput property of the SpheroidStandard component.

<logical> energyGasInputIsGettable() Returns true if the energyGasInput property is gettable for the spheroid component class.

<logical> energyGasInputIsSettable() Specify whether the energyGasInput property of the spheroid component is settable.

<void> energyGasInputRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interrupt [interruptProcedure]↔) Accumulate the rate of change of the energyGasInput property of the standard implementation of the spheroid component using a deferred function.

<void> energyGasInputRateFunction() Set the function to be used for the rate method of the energyGasInput property of the SpheroidStandard component.

<double precision> energyGasInputRateGet() Returns a zero rate for the energyGasInput property for the spheroid component class.

<logical> energyGasInputRateIsAttached() Return true if the deferred function used to rate the energyGasInput property of the SpheroidStandard component class has been attached.

<double precision> halfMassRadius() Returns the default value for the halfMassRadius property for the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> halfMassRadiusAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spheroid class that have the desired attributes for the halfMassRadius property

<logical> halfMassRadiusIsGettable() Returns true if the halfMassRadius property is gettable for the spheroid component class.

<logical> halfMassRadiusIsSettable() Specify whether the halfMassRadius property of the spheroid component is settable.

<double precision> halfMassRadiusRateGet() Returns a zero rate for the halfMassRadius property for the spheroid component class.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a standard member of the spheroid component.

<logical> isInitialized() Get the isInitialized property of an standard implementation of the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> isInitializedAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spheroid class that have the desired attributes for the isInitialized property

<logical> isInitializedIsGettable() Returns true if the isInitialized property is gettable for the spheroid component class.

<logical> isInitializedIsSettable() Specify whether the isInitialized property of the spheroid component is settable.

<logical> isInitializedRateGet() Returns a zero rate for the isInitialized property for the spheroid component class.

<void> isInitializedSet(<logical> setValue→) Set the isInitialized property of an standard implementation of the spheroid component class.

<type(stellarLuminosities)> luminositiesStellar() Get the luminositiesStellar property of an standard implementation of the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> luminositiesStellarAttributeMatch(<logica [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spheroid class that have the desired attributes for the luminositiesStellar property

<integer> luminositiesStellarCount() Return a count of the number of scalar properties in the luminositiesStellar property of an standard implementation of the spheroid component class.

<void> luminositiesStellarInactive() Indicate that the luminositiesStellar property of an standard implementation of the spheroid component class is inactive for differential equation solving.

<logical> luminositiesStellarIsGettable() Returns true if the luminositiesStellar property is gettable for the spheroid component class.

<logical> luminositiesStellarIsSettable() Specify whether the luminositiesStellar property of the spheroid component is settable.

<void> luminositiesStellarRate(<type(stellarLuminosities)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the luminositiesStellar property of an standard implementation of the spheroid component class.

<type(stellarLuminosities)> luminositiesStellarRateGet() Get the rate of change of the luminositiesStellar property of an standard implementation of the spheroid component class.

<void> luminositiesStellarScale(<type(stellarLuminosities)> setValue→) Set the absolute scale of the luminositiesStellar property of an standard implementation of the spheroid component class.

<void> luminositiesStellarSet(<type(stellarLuminosities)> setValue→) Set the luminositiesStellar property of an standard implementation of the spheroid component class.

<double precision> massGas() Get the massGas property of an standard implementation of the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> massGasAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spheroid class that have the desired attributes for the massGas property

<integer> massGasCount() Return a count of the number of scalar properties in the massGas property of an standard implementation of the spheroid component class.

<void> massGasInactive() Indicate that the massGas property of an standard implementation of the spheroid component class is inactive for differential equation solving.

<logical> massGasIsGettable() Returns true if the massGas property is gettable for the spheroid component class.

<logical> massGasIsSettable() Specify whether the massGas property of the spheroid component is settable.

<void> massGasRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the massGas property of an standard implementation of the spheroid component class.

<double precision> massGasRateGet() Get the rate of change of the massGas property of an standard implementation of the spheroid component class.

<void> massGasScale(<double precision> setValue→) Set the absolute scale of the massGas property of an standard implementation of the spheroid component class.

<void> massGasSet(<double precision> setValue→) Set the massGas property of an standard implementation of the spheroid component class.

<double precision> massGasSink() Returns the default value for the massGasSink property for the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> massGasSinkAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spheroid class that have the desired attributes for the massGasSink property

<integer> massGasSinkCount() Compute the count of evolvable quantities in the massGasSink property of the SpheroidStandard component.

<logical> massGasSinkIsGettable() Returns true if the massGasSink property is gettable for the spheroid component class.

<logical> massGasSinkIsSettable() Specify whether the massGasSink property of the spheroid component is settable.

<void> massGasSinkRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTas [interruptProcedure]↔) Accumulate the rate of change of the massGasSink property of the standard implementation of the spheroid component using a deferred function.

<void> massGasSinkRateFunction() Set the function to be used for the rate method of the massGasSink property of the SpheroidStandard component.

<double precision> massGasSinkRateGet() Returns a zero rate for the massGasSink property for the spheroid component class.

<logical> massGasSinkRateIsAttached() Return true if the deferred function used to rate the massGasSink property of the SpheroidStandard component class has been attached.

<double precision> massStellar() Get the massStellar property of an standard implementation of the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> massStellarAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spheroid class that have the desired attributes for the massStellar property

<integer> massStellarCount() Return a count of the number of scalar properties in the massStellar property of an standard implementation of the spheroid component class.

<double precision> massStellarFormed() Get the massStellarFormed property of an standard implementation of the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> massStellarFormedAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spheroid class that have the desired attributes for the massStellarFormed property

<integer> massStellarFormedCount() Return a count of the number of scalar properties in the massStellarFormed property of an standard implementation of the spheroid component class.

<void> massStellarFormedInactive() Indicate that the massStellarFormed property of an standard implementation of the spheroid component class is inactive for differential equation solving.

<logical> massStellarFormedIsGettable() Returns true if the massStellarFormed property is gettable for the spheroid component class.

<logical> massStellarFormedIsSettable() Specify whether the massStellarFormed property of the spheroid component is settable.

<void> massStellarFormedRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interr [interruptProcedure]↔) Accumulate to the rate of change of the massStellarFormed property of an standard implementation of the spheroid component class.

<double precision> massStellarFormedRateGet() Get the rate of change of the massStellarFormed property of an standard implementation of the spheroid component class.

<void> massStellarFormedScale(<double precision> setValue→) Set the absolute scale of the massStellarFormed property of an standard implementation of the spheroid component class.

<void> massStellarFormedSet(<double precision> setValue→) Set the massStellarFormed property of an standard implementation of the spheroid component class.

<void> massStellarInactive() Indicate that the massStellar property of an standard implementation of the spheroid component class is inactive for differential equation solving.

<logical> massStellarIsGettable() Returns true if the massStellar property is gettable for the spheroid component class.

<logical> massStellarIsSettable() Specify whether the massStellar property of the spheroid component is settable.

<void> massStellarRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTas [interruptProcedure]↔) Accumulate to the rate of change of the massStellar property of an standard implementation of the spheroid component class.

<double precision> massStellarRateGet() Get the rate of change of the massStellar property of an standard implementation of the spheroid component class.

<void> massStellarScale(<double precision> setValue→) Set the absolute scale of the massStellar property of an standard implementation of the spheroid component class.

<void> massStellarSet(<double precision> setValue→) Set the massStellar property of an standard implementation of the spheroid component class.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a standard implementation of the spheroid component class.

<logical> nullIsActive() Return true if the null implementation of the spheroid component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a spheroid component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a standard implementation of the spheroid component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a standard implementation of the spheroid component.

<void> postOutput(<double precision> time→) Perform post-output processing for a standard implementation of the spheroid component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<double precision> radius() Get the radius property of an standard implementation of the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> radiusAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spheroid class that have the desired attributes for the radius property

<logical> radiusIsGettable() Returns true if the radius property is gettable for the spheroid component class.

<logical> radiusIsSettable() Specify whether the radius property of the spheroid component is settable.

<double precision> radiusRateGet() Returns a zero rate for the radius property for the spheroid component class.

<void> radiusSet(<double precision> setValue→) Set the radius property of an standard implementation of the spheroid component class.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a standard implementation of the spheroid component.

<void> serializeASCII() Serialize the contents of a standard implementation of the spheroid component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a standard implementation of the spheroid component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a standard implementation of the spheroid component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a standard implementation of the spheroid component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a standard implementation of the spheroid component to XML.

<integer(c_size_t)> sizeOf() Return the size in bytes of a standard implementation of the spheroid component.

<logical> standardIsActive() Return true if the standard implementation of the spheroid component is the active choice.

<type(history)> starFormationHistory() Get the starFormationHistory property of an standard implementation of the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> starFormationHistoryAttributeMatch(<logic
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the spheroid class that have the desired attributes for the starFormationHistory property

<integer> starFormationHistoryCount() Return a count of the number of scalar properties in the starFormationHistory property of an standard implementation of the spheroid component class.

<void> starFormationHistoryInactive() Indicate that the starFormationHistory property of an standard implementation of the spheroid component class is inactive for differential equation solving.

<logical> starFormationHistoryIsGettable() Returns true if the starFormationHistory property is gettable for the spheroid component class.

<logical> starFormationHistoryIsSettable() Specify whether the starFormationHistory property of the spheroid component is settable.

<void> starFormationHistoryRate(<type(history)> setValue→, <logical> [interrupt]↔, <*procedure(interr
[interruptProcedure]↔) Accumulate the rate of change of the starFormationHistory property of the standard implementation of the spheroid component using a deferred function.

<void> starFormationHistoryRateFunction() Set the function to be used for the rate method of the starFormationHistory property of the SpheroidStandard component.

<type(history)> starFormationHistoryRateGet() Get the rate of change of the starFormationHistory property of an standard implementation of the spheroid component class.

<void> starFormationHistoryRateIntrinsic(<type(history)> setValue→, <logical> [interrupt]↔,
<*procedure(interruptTask)> [interruptProcedure]↔) Accumulate directly (i.e. circumventing any deferred function binding) to the rate of change of the starFormationHistory property of an standard implementation of the spheroid component class.

<logical> starFormationHistoryRateIsAttached() Return true if the deferred function used to rate the starFormationHistory property of the SpheroidStandard component class has been attached.

<void> starFormationHistoryScale(<type(history)> setValue→) Set the absolute scale of the starFormationHistory property of an standard implementation of the spheroid component class.

<void> starFormationHistorySet(<type(history)> setValue→) Set the starFormationHistory property of an standard implementation of the spheroid component class.

<double precision> starFormationRate() Get the value of the starFormationRate property of the standard implementation of the spheroid component using a deferred function.

<type(varying_string), allocatable, dimension(:) => matches> starFormationRateAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spheroid class that have the desired attributes for the starFormationRate property

<void> starFormationRateFunction() Set the function to be used for the get method of the starFormationRate property of the SpheroidStandard component.

<logical> starFormationRateIsAttached() Return true if the deferred function used to get the starFormationRate property of the SpheroidStandard component class has been attached.

<logical> starFormationRateIsGettable() Returns true if the starFormationRate property is gettable for the spheroid component class.

<logical> starFormationRateIsSettable() Specify whether the starFormationRate property of the spheroid component is settable.

<double precision> starFormationRateRateGet() Returns a zero rate for the starFormationRate property for the spheroid component class.

<type(history)> stellarPropertiesHistory() Get the stellarPropertiesHistory property of an standard implementation of the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> stellarPropertiesHistoryAttributeMatch(<l [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spheroid class that have the desired attributes for the stellarPropertiesHistory property

<integer> stellarPropertiesHistoryCount() Return a count of the number of scalar properties in the stellarPropertiesHistory property of an standard implementation of the spheroid component class.

<void> stellarPropertiesHistoryInactive() Indicate that the stellarPropertiesHistory property of an standard implementation of the spheroid component class is inactive for differential equation solving.

<logical> stellarPropertiesHistoryIsGettable() Returns true if the stellarPropertiesHistory property is gettable for the spheroid component class.

<logical> stellarPropertiesHistoryIsSettable() Specify whether the stellarPropertiesHistory property of the spheroid component is settable.

<void> stellarPropertiesHistoryRate(<type(history)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the stellarPropertiesHistory property of the standard implementation of the spheroid component using a deferred function.

<void> stellarPropertiesHistoryRateFunction() Set the function to be used for the rate method of the stellarPropertiesHistory property of the SpheroidStandard component.

<type(history)> stellarPropertiesHistoryRateGet() Get the rate of change of the stellarPropertiesHistory property of an standard implementation of the spheroid component class.

<void> stellarPropertiesHistoryRateIntrinsic(<type(history)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate directly (i.e. circumventing any deferred function binding) to the rate of change of the stellarPropertiesHistory property of an standard implementation of the spheroid component class.

<logical> stellarPropertiesHistoryRateIsAttached() Return true if the deferred function used to rate the stellarPropertiesHistory property of the SpheroidStandard component class has been attached.

<void> stellarPropertiesHistoryScale(<type(history)> setValue→) Set the absolute scale of the stellarPropertiesHistory property of an standard implementation of the spheroid component class.

<void> stellarPropertiesHistorySet(<type(history)> setValue→) Set the stellarPropertiesHistory property of an standard implementation of the spheroid component class.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<type(varying_string)> type() Returns the type name for the standard implementation of the spheroid component class.

<double precision> velocity() Get the velocity property of an standard implementation of the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> velocityAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spheroid class that have the desired attributes for the velocity property

<logical> velocityIsGettable() Returns true if the velocity property is gettable for the spheroid component class.

<logical> velocityIsSettable() Specify whether the velocity property of the spheroid component is settable.

<double precision> velocityRateGet() Returns a zero rate for the velocity property for the spheroid component class.

<void> velocitySet(<double precision> setValue→) Set the velocity property of an standard implementation of the spheroid component class.

<logical> verySimpleIsActive() Return true if the verySimple implementation of the spheroid component is the active choice.

nodeComponentSpheroidVerySimple

<type(abundances)> abundancesGas() Get the abundancesGas property of an verySimple implementation of the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> abundancesGasAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spheroid class that have the desired attributes for the abundancesGas property

<integer> abundancesGasCount() Return a count of the number of scalar properties in the abundancesGas property of an verySimple implementation of the spheroid component class.

<void> abundancesGasInactive() Indicate that the abundancesGas property of an verySimple implementation of the spheroid component class is inactive for differential equation solving.

<logical> abundancesGasIsGettable() Returns true if the abundancesGas property is gettable for the spheroid component class.

<logical> abundancesGasIsSettable() Specify whether the abundancesGas property of the spheroid component is settable.

<void> abundancesGasRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptT [interruptProcedure]↔) Accumulate to the rate of change of the abundancesGas property of an verySimple implementation of the spheroid component class.

<type(abundances)> abundancesGasRateGet() Get the rate of change of the abundancesGas property of an verySimple implementation of the spheroid component class.

<void> abundancesGasScale(<type(abundances)> setValue→) Set the absolute scale of the abundancesGas property of an verySimple implementation of the spheroid component class.

<void> abundancesGasSet(<type(abundances)> setValue→) Set the abundancesGas property of an verySimple implementation of the spheroid component class.

<type(abundances)> abundancesStellar() Get the abundancesStellar property of an verySimple implementation of the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> abundancesStellarAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spheroid class that have the desired attributes for the abundancesStellar property

<integer> abundancesStellarCount() Return a count of the number of scalar properties in the abundancesStellar property of an verySimple implementation of the spheroid component class.

<void> abundancesStellarInactive() Indicate that the abundancesStellar property of an verySimple implementation of the spheroid component class is inactive for differential equation solving.

<logical> abundancesStellarIsGettable() Returns true if the abundancesStellar property is gettable for the spheroid component class.

<logical> abundancesStellarIsSettable() Specify whether the abundancesStellar property of the spheroid component is settable.

<void> abundancesStellarRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interr
[interruptProcedure]↔) Accumulate to the rate of change of the abundancesStellar property of an verySimple implementation of the spheroid component class.

<type(abundances)> abundancesStellarRateGet() Get the rate of change of the abundancesStellar property of an verySimple implementation of the spheroid component class.

<void> abundancesStellarScale(<type(abundances)> setValue→) Set the absolute scale of the abundancesStellar property of an verySimple implementation of the spheroid component class.

<void> abundancesStellarSet(<type(abundances)> setValue→) Set the abundancesStellar property of an verySimple implementation of the spheroid component class.

<double precision> angularMomentum() Returns the default value for the angularMomentum property for the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> angularMomentumAttributeMatch(<logical>
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the spheroid class that have the desired attributes for the angularMomentum property

<integer> angularMomentumCount() Compute the count of evolvable quantities in the angularMomentum property of the SpheroidStandard component.

<logical> angularMomentumIsGettable() Returns true if the angularMomentum property is gettable for the spheroid component class.

<logical> angularMomentumIsSettable() Specify whether the angularMomentum property of the spheroid component is settable.

<void> angularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interrup
[interruptProcedure]↔) Accept a rate set for the angularMomentum property of the spheroid component class. Trigger an interrupt to create the component.

<double precision> angularMomentumRateGet() Returns a zero rate for the angularMomentum property for the spheroid component class.

<void> angularMomentumScale(<double precision> value) Set the scale of the angularMomentum property of the SpheroidStandard component.

<void> angularMomentumSet(<double precision> value) Set the angularMomentum property of the spheroid component.

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a verySimple implementation of the spheroid component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType>
[massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a verySimple implementation of the spheroid component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a verySimple implementation of the spheroid component from array.

<void> destroy() Finalize a verySimple implementation of the spheroid component.

<void> dumpASCII() Dump the content of a spheroid component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<double precision> energyGasInput() Returns the default value for the energyGasInput property for the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> energyGasInputAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spheroid class that have the desired attributes for the energyGasInput property

<integer> energyGasInputCount() Compute the count of evolvable quantities in the energyGasInput property of the SpheroidStandard component.

<logical> energyGasInputIsGettable() Returns true if the energyGasInput property is gettable for the spheroid component class.

<logical> energyGasInputIsSettable() Specify whether the energyGasInput property of the spheroid component is settable.

<void> energyGasInputRate(<double precision> value) Cumulate to the rate of the energyGasInput property of the SpheroidStandard component.

<double precision> energyGasInputRateGet() Returns a zero rate for the energyGasInput property for the spheroid component class.

<double precision> halfMassRadius() Returns the default value for the halfMassRadius property for the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> halfMassRadiusAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spheroid class that have the desired attributes for the halfMassRadius property

<logical> halfMassRadiusIsGettable() Returns true if the halfMassRadius property is gettable for the spheroid component class.

<logical> halfMassRadiusIsSettable() Specify whether the halfMassRadius property of the spheroid component is settable.

<double precision> halfMassRadiusRateGet() Returns a zero rate for the halfMassRadius property for the spheroid component class.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a verySimple member of the spheroid component.

<logical> isInitialized() Get the isInitialized property of an verySimple implementation of the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> isInitializedAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spheroid class that have the desired attributes for the isInitialized property

<logical> isInitializedIsGettable() Returns true if the isInitialized property is gettable for the spheroid component class.

<logical> isInitializedIsSettable() Specify whether the isInitialized property of the spheroid component is settable.

<logical> isInitializedRateGet() Returns a zero rate for the isInitialized property for the spheroid component class.

<void> isInitializedSet(<logical> setValue→) Set the isInitialized property of an verySimple implementation of the spheroid component class.

<type(stellarLuminosities)> luminositiesStellar() Get the luminositiesStellar property of an verySimple implementation of the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> luminositiesStellarAttributeMatch(<logica [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spheroid class that have the desired attributes for the luminositiesStellar property

<integer> luminositiesStellarCount() Return a count of the number of scalar properties in the luminositiesStellar property of an verySimple implementation of the spheroid component class.

<void> luminositiesStellarInactive() Indicate that the luminositiesStellar property of an verySimple implementation of the spheroid component class is inactive for differential equation solving.

<logical> luminositiesStellarIsGettable() Returns true if the luminositiesStellar property is gettable for the spheroid component class.

<logical> luminositiesStellarIsSettable() Specify whether the luminositiesStellar property of the spheroid component is settable.

<void> luminositiesStellarRate(<type(stellarLuminosities)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the luminositiesStellar property of an verySimple implementation of the spheroid component class.

<type(stellarLuminosities)> luminositiesStellarRateGet() Get the rate of change of the luminositiesStellar property of an verySimple implementation of the spheroid component class.

<void> luminositiesStellarScale(<type(stellarLuminosities)> setValue→) Set the absolute scale of the luminositiesStellar property of an verySimple implementation of the spheroid component class.

<void> luminositiesStellarSet(<type(stellarLuminosities)> setValue→) Set the luminositiesStellar property of an verySimple implementation of the spheroid component class.

<double precision> massGas() Get the massGas property of an verySimple implementation of the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> massGasAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spheroid class that have the desired attributes for the massGas property

<integer> massGasCount() Return a count of the number of scalar properties in the massGas property of an verySimple implementation of the spheroid component class.

<void> massGasInactive() Indicate that the massGas property of an verySimple implementation of the spheroid component class is inactive for differential equation solving.

<logical> massGasIsGettable() Returns true if the massGas property is gettable for the spheroid component class.

<logical> massGasIsSettable() Specify whether the massGas property of the spheroid component is settable.

<void> massGasRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the massGas property of an verySimple implementation of the spheroid component class.

<double precision> massGasRateGet() Get the rate of change of the massGas property of an verySimple implementation of the spheroid component class.

<void> massGasScale(<double precision> setValue→) Set the absolute scale of the massGas property of an verySimple implementation of the spheroid component class.

<void> massGasSet(<double precision> setValue→) Set the massGas property of an verySimple implementation of the spheroid component class.

<double precision> massGasSink() Returns the default value for the massGasSink property for the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> massGasSinkAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spheroid class that have the desired attributes for the massGasSink property

<integer> massGasSinkCount() Compute the count of evolvable quantities in the massGasSink property of the SpheroidStandard component.

<logical> massGasSinkIsGettable() Returns true if the massGasSink property is gettable for the spheroid component class.

<logical> massGasSinkIsSettable() Specify whether the massGasSink property of the spheroid component is settable.

<void> massGasSinkRate(<double precision> value) Cumulate to the rate of the massGasSink property of the SpheroidStandard component.

<double precision> massGasSinkRateGet() Returns a zero rate for the massGasSink property for the spheroid component class.

<double precision> massStellar() Get the massStellar property of an verySimple implementation of the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> massStellarAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spheroid class that have the desired attributes for the massStellar property

<integer> massStellarCount() Return a count of the number of scalar properties in the massStellar property of an verySimple implementation of the spheroid component class.

<double precision> massStellarFormed() Returns the default value for the massStellarFormed property for the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> massStellarFormedAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spheroid class that have the desired attributes for the massStellarFormed property

<integer> massStellarFormedCount() Compute the count of evolvable quantities in the massStellarFormed property of the SpheroidStandard component.

<logical> massStellarFormedIsGettable() Returns true if the massStellarFormed property is gettable for the spheroid component class.

<logical> massStellarFormedIsSettable() Specify whether the massStellarFormed property of the spheroid component is settable.

<void> massStellarFormedRate(<double precision> value) Cumulate to the rate of the massStellarFormed property of the SpheroidStandard component.

<double precision> massStellarFormedRateGet() Returns a zero rate for the massStellarFormed property for the spheroid component class.

<void> massStellarFormedScale(<double precision> value) Set the scale of the massStellarFormed property of the SpheroidStandard component.

<void> massStellarFormedSet(<double precision> value) Set the massStellarFormed property of the spheroid component.

<void> massStellarInactive() Indicate that the massStellar property of an verySimple implementation of the spheroid component class is inactive for differential equation solving.

<logical> massStellarIsGettable() Returns true if the massStellar property is gettable for the spheroid component class.

<logical> massStellarIsSettable() Specify whether the massStellar property of the spheroid component is settable.

<void> massStellarRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTas [interruptProcedure]↔) Accumulate to the rate of change of the massStellar property of an verySimple implementation of the spheroid component class.

<double precision> massStellarRateGet() Get the rate of change of the massStellar property of an verySimple implementation of the spheroid component class.

<void> massStellarScale(<double precision> setValue→) Set the absolute scale of the massStellar property of an verySimple implementation of the spheroid component class.

<void> massStellarSet(<double precision> setValue→) Set the massStellar property of an verySimple implementation of the spheroid component class.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a verySimple implementation of the spheroid component class.

<logical> nullIsActive() Return true if the null implementation of the spheroid component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_- int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a spheroid component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a verySimple implementation of the spheroid component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a verySimple implementation of the spheroid component.

<void> postOutput(<double precision> time→) Perform post-output processing for a verySimple implementation of the spheroid component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<double precision> radius() Get the radius property of an verySimple implementation of the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> radiusAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spheroid class that have the desired attributes for the radius property

<logical> radiusIsGettable() Returns true if the radius property is gettable for the spheroid component class.

<logical> radiusIsSettable() Specify whether the radius property of the spheroid component is settable.

<double precision> radiusRateGet() Returns a zero rate for the radius property for the spheroid component class.

<void> radiusSet(<double precision> setValue→) Set the radius property of an verySimple implementation of the spheroid component class.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a verySimple implementation of the spheroid component.

<void> serializeASCII() Serialize the contents of a verySimple implementation of the spheroid component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a verySimple implementation of the spheroid component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a verySimple implementation of the spheroid component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a verySimple implementation of the spheroid component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a verySimple implementation of the spheroid component to XML.

<integer(c_size_t)> sizeOf() Return the size in bytes of a verySimple implementation of the spheroid component.

<logical> standardIsActive() Return true if the standard implementation of the spheroid component is the active choice.

<type(history)> starFormationHistory() Returns the default value for the starFormationHistory property for the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> starFormationHistoryAttributeMatch(<logic
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the spheroid class that have the desired attributes for the starFormationHistory property

<integer> starFormationHistoryCount() Compute the count of evolvable quantities in the starFormationHistory property of the SpheroidStandard component.

<logical> starFormationHistoryIsGettable() Returns true if the starFormationHistory property is gettable for the spheroid component class.

<logical> starFormationHistoryIsSettable() Specify whether the starFormationHistory property of the spheroid component is settable.

<void> starFormationHistoryRate(<type(history)> setValue→, <logical> [interrupt]↔, <*procedure(interr
[interruptProcedure]↔) Accept a rate set for the starFormationHistory property of the spheroid component class. Trigger an interrupt to create the component.

<type(history)> starFormationHistoryRateGet() Returns a zero rate for the starFormationHistory property for the spheroid component class.

<void> starFormationHistoryScale(<type(history)> value) Set the scale of the starFormationHistory property of the SpheroidStandard component.

<void> starFormationHistorySet(<type(history)> value) Set the starFormationHistory property of the spheroid component.

<double precision> starFormationRate() Get the value of the starFormationRate property of the verySimple implementation of the spheroid component using a deferred function.

<type(varying_string), allocatable, dimension(:) => matches> starFormationRateAttributeMatch(<logical>
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the spheroid class that have the desired attributes for the starFormationRate property

<void> starFormationRateFunction() Set the function to be used for the get method of the starFormationRate property of the SpheroidVerySimple component.

<logical> starFormationRateIsAttached() Return true if the deferred function used to get the starFormationRate property of the SpheroidVerySimple component class has been attached.

<logical> starFormationRateIsGettable() Returns true if the starFormationRate property is gettable for the spheroid component class.

<logical> starFormationRateIsSettable() Specify whether the starFormationRate property of the spheroid component is settable.

<double precision> starFormationRateRateGet() Returns a zero rate for the starFormationRate property for the spheroid component class.

<type(history)> stellarPropertiesHistory() Get the stellarPropertiesHistory property of an verySimple implementation of the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> stellarPropertiesHistoryAttributeMatch(<l
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the spheroid class that have the desired attributes for the stellarPropertiesHistory property

<integer> stellarPropertiesHistoryCount() Return a count of the number of scalar properties in the stellarPropertiesHistory property of an verySimple implementation of the spheroid component class.

<void> stellarPropertiesHistoryInactive() Indicate that the stellarPropertiesHistory property of an verySimple implementation of the spheroid component class is inactive for differential equation solving.

<logical> stellarPropertiesHistoryIsGettable() Returns true if the stellarPropertiesHistory property is gettable for the spheroid component class.

<logical> stellarPropertiesHistoryIsSettable() Specify whether the stellarPropertiesHistory property of the spheroid component is settable.

<void> stellarPropertiesHistoryRate(<type(history)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the stellarPropertiesHistory property of an verySimple implementation of the spheroid component class.

<type(history)> stellarPropertiesHistoryRateGet() Get the rate of change of the stellarPropertiesHistory property of an verySimple implementation of the spheroid component class.

<void> stellarPropertiesHistoryScale(<type(history)> setValue→) Set the absolute scale of the stellarPropertiesHistory property of an verySimple implementation of the spheroid component class.

<void> stellarPropertiesHistorySet(<type(history)> setValue→) Set the stellarPropertiesHistory property of an verySimple implementation of the spheroid component class.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<type(varying_string)> type() Returns the type name for the verySimple implementation of the spheroid component class.

<double precision> velocity() Get the velocity property of an verySimple implementation of the spheroid component class.

<type(varying_string), allocatable, dimension(:) => matches> velocityAttributeMatch(<logical>
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the spheroid class that have the desired attributes for the velocity property

<logical> velocityIsGettable() Returns true if the velocity property is gettable for the spheroid component class.

<logical> velocityIsSettable() Specify whether the velocity property of the spheroid component is settable.

<double precision> velocityRateGet() Returns a zero rate for the velocity property for the spheroid component class.

<void> velocitySet(<double precision> setValue→) Set the velocity property of an verySimple implementation of the spheroid component class.

<logical> verySimpleIsActive() Return true if the verySimple implementation of the spheroid component is the active choice.

nodeComponentSpin

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a generic spin component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Read properties from raw file.

<void> deserializeValues(<double(:)> array→, <integer> propertyType→) Deserialize the evolvable quantities from an array.

<void> destroy() Finalize a generic spin component.

<void> dumpASCII() Dump the content of a spin component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a generic spin component.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a nodeComponent object.

<logical> nullIsActive() Return true if the null implementation of the spin component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a spin component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a generic spin component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Establish the names of properties to output for a generic spin component.

<void> postOutput(<double precision> time→) Perform post-output processing of a spin component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<logical> preset3DIsActive() Return true if the preset3D implementation of the spin component is the active choice.

<logical> presetIsActive() Return true if the preset implementation of the spin component is the active choice.

<logical> randomIsActive() Return true if the random implementation of the spin component is the active choice.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets() Set offsets into serialization arrays.

<void> serializeASCII() Serialize the content of a spin component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the number of evolvable quantities to be evolved.

<void> serializeRaw(<integer> fileHandle→) Generate a binary dump of all properties.

<void> serializeValues(<double(:)> array←, <integer> propertyType→) Serialize the evolvable quantities to an array.

<void> serializeXML() Generate an XML dump of all properties.

<integer(c_size_t)> sizeOf() Return the size in bytes of a nodeComponentSpin component.

<double precision> spin() Returns the default value for the spin property for the spin component class.

<type(varying_string), allocatable, dimension(:) => matches> spinAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spin class that have the desired attributes for the spin property

<integer> spinCount() Compute the count of evolvable quantities in the spin property of the SpinPreset component.

<double precision> spinGrowthRate() Returns the default value for the spinGrowthRate property for the spin component class.

<type(varying_string), allocatable, dimension(:) => matches> spinGrowthRateAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spin class that have the desired attributes for the spinGrowthRate property

<logical> spinGrowthRateIsGettable() Returns true if the spinGrowthRate property is gettable for the spin component class.

<logical> spinGrowthRateIsSettable() Specify whether the spinGrowthRate property of the spin component is settable.

<double precision> spinGrowthRateRateGet() Returns a zero rate for the spinGrowthRate property for the spin component class.

<void> spinGrowthRateSet(<double precision> value) Set the spinGrowthRate property of the spin component.

<logical> spinIsGettable() Returns true if the spin property is gettable for the spin component class.

<logical> spinIsSettable() Specify whether the spin property of the spin component is settable.

<void> spinRate(<double precision> value) Cumulate to the rate of the spin property of the SpinPreset component.

<double precision> spinRateGet() Returns a zero rate for the spin property for the spin component class.

<void> spinScale(<double precision> value) Set the scale of the spin property of the SpinPreset component.

<void> spinSet(<double precision> value) Set the spin property of the spin component.

<double precision, dimension(:), allocatable => classDefault> spinVector() Returns the default value for the spinVector property for the spin component class.

<type(varying_string), allocatable, dimension(:) => matches> spinVectorAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spin class that have the desired attributes for the spinVector property

<integer> spinVectorCount() Compute the count of evolvable quantities in the spinVector property of the SpinVitvitska component.

<double precision, dimension(:), allocatable => classDefault> spinVectorGrowthRate() Returns the default value for the spinVectorGrowthRate property for the spin component class.

<type(varying_string), allocatable, dimension(:) => matches> spinVectorGrowthRateAttributeMatch(<logic [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spin class that have the desired attributes for the spinVectorGrowthRate property

<logical> spinVectorGrowthRateIsGettable() Returns true if the spinVectorGrowthRate property is gettable for the spin component class.

<logical> spinVectorGrowthRateIsSettable() Specify whether the spinVectorGrowthRate property of the spin component is settable.

<double precision, dimension(:), allocatable => classDefault> spinVectorGrowthRateRateGet() Returns a zero rate for the spinVectorGrowthRate property for the spin component class.

<void> spinVectorGrowthRateSet(<double precision[:]> value) Set the spinVectorGrowthRate property of the spin component.

<logical> spinVectorIsGettable() Returns true if the spinVector property is gettable for the spin component class.

<logical> spinVectorIsSettable() Specify whether the spinVector property of the spin component is settable.

<void> spinVectorRate(<double precision[:]> value) Cumulate to the rate of the spinVector property of the SpinVitvitska component.

<double precision, dimension(:), allocatable => classDefault> spinVectorRateGet() Returns a zero rate for the spinVector property for the spin component class.

<void> spinVectorScale(<double precision[:]> value) Set the scale of the spinVector property of the SpinVitvitska component.

<void> spinVectorSet(<double precision[:]> value) Set the spinVector property of the spin component.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<type(varying_string)> type() Returns the type name for the spin component class.

<logical> vitvitskaIsActive() Return true if the vitvitska implementation of the spin component is the active choice.

nodeComponentSpinNull

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a null implementation of the spin component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a null implementation of the spin component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a null implementation of the spin component from array.

<void> destroy() Finalize a null implementation of the spin component.

<void> dumpASCII() Dump the content of a spin component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a null member of the spin component.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a null implementation of the spin component class.

<logical> nullIsActive() Return true if the null implementation of the spin component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a spin component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a null implementation of the spin component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a null implementation of the spin component.

<void> postOutput(<double precision> time→) Perform post-output processing of a spin component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<logical> preset3DIsActive() Return true if the preset3D implementation of the spin component is the active choice.

<logical> presetIsActive() Return true if the preset implementation of the spin component is the active choice.

<logical> randomIsActive() Return true if the random implementation of the spin component is the active choice.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a null implementation of the spin component.

<void> serializeASCII() Serialize the contents of a null implementation of the spin component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a null implementation of the spin component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a null implementation of the spin component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a null implementation of the spin component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a null implementation of the spin component to XML.

<integer(c_size_t)> sizeOf() Return the size in bytes of a null implementation of the spin component.

<double precision> spin() Returns the default value for the spin property for the spin component class.

<type(varying_string), allocatable, dimension(:) => matches> spinAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spin class that have the desired attributes for the spin property

<integer> spinCount() Compute the count of evolvable quantities in the spin property of the SpinPreset component.

<double precision> spinGrowthRate() Returns the default value for the spinGrowthRate property for the spin component class.

<type(varying_string), allocatable, dimension(:) => matches> spinGrowthRateAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spin class that have the desired attributes for the spinGrowthRate property

<logical> spinGrowthRateIsGettable() Returns true if the spinGrowthRate property is gettable for the spin component class.

<logical> spinGrowthRateIsSettable() Specify whether the spinGrowthRate property of the spin component is settable.

<double precision> spinGrowthRateRateGet() Returns a zero rate for the spinGrowthRate property for the spin component class.

<void> spinGrowthRateSet(<double precision> value) Set the spinGrowthRate property of the spin component.

<logical> spinIsGettable() Returns true if the spin property is gettable for the spin component class.

<logical> spinIsSettable() Specify whether the spin property of the spin component is settable.

<void> spinRate(<double precision> value) Cumulate to the rate of the spin property of the SpinPreset component.

<double precision> spinRateGet() Returns a zero rate for the spin property for the spin component class.

<void> spinScale(<double precision> value) Set the scale of the spin property of the SpinPreset component.

<void> spinSet(<double precision> value) Set the spin property of the spin component.

<double precision, dimension(:), allocatable => classDefault> spinVector() Returns the default value for the spinVector property for the spin component class.

<type(varying_string), allocatable, dimension(:) => matches> spinVectorAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spin class that have the desired attributes for the spinVector property

<integer> spinVectorCount() Compute the count of evolvable quantities in the spinVector property of the SpinVitvitska component.

<double precision, dimension(:), allocatable => classDefault> spinVectorGrowthRate() Returns the default value for the spinVectorGrowthRate property for the spin component class.

<type(varying_string), allocatable, dimension(:) => matches> spinVectorGrowthRateAttributeMatch(<logic
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the spin class that have the desired attributes for the spinVectorGrowthRate property

<logical> spinVectorGrowthRateIsGettable() Returns true if the spinVectorGrowthRate property is gettable for the spin component class.

<logical> spinVectorGrowthRateIsSettable() Specify whether the spinVectorGrowthRate property of the spin component is settable.

<double precision, dimension(:), allocatable => classDefault> spinVectorGrowthRateRateGet()
Returns a zero rate for the spinVectorGrowthRate property for the spin component class.

<void> spinVectorGrowthRateSet(<double precision[:]> value) Set the spinVectorGrowthRate property of the spin component.

<logical> spinVectorIsGettable() Returns true if the spinVector property is gettable for the spin component class.

<logical> spinVectorIsSettable() Specify whether the spinVector property of the spin component is settable.

<void> spinVectorRate(<double precision[:]> value) Cumulate to the rate of the spinVector property of the SpinVitvitska component.

<double precision, dimension(:), allocatable => classDefault> spinVectorRateGet() Returns a zero rate for the spinVector property for the spin component class.

<void> spinVectorScale(<double precision[:]> value) Set the scale of the spinVector property of the SpinVitvitska component.

<void> spinVectorSet(<double precision[:]> value) Set the spinVector property of the spin component.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→,
<massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<type(varying_string)> type() Returns the type name for the null implementation of the spin component class.

<logical> vitvitskaIsActive() Return true if the vitvitska implementation of the spin component is the active choice.

nodeComponentSpinPreset

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a preset implementation of the spin component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a preset implementation of the spin component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a preset implementation of the spin component from array.

<void> destroy() Finalize a preset implementation of the spin component.

<void> dumpASCII() Dump the content of a spin component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a preset member of the spin component.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a preset implementation of the spin component class.

<logical> nullIsActive() Return true if the null implementation of the spin component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a spin component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a preset implementation of the spin component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a preset implementation of the spin component.

<void> postOutput(<double precision> time→) Perform post-output processing of a spin component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<logical> preset3DIsActive() Return true if the preset3D implementation of the spin component is the active choice.

<logical> presetIsActive() Return true if the preset implementation of the spin component is the active choice.

<logical> randomIsActive() Return true if the random implementation of the spin component is the active choice.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a preset implementation of the spin component.

<void> serializeASCII() Serialize the contents of a preset implementation of the spin component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a preset implementation of the spin component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a preset implementation of the spin component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a preset implementation of the spin component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a preset implementation of the spin component to XML.

<integer(c_size_t)> sizeOf() Return the size in bytes of a preset implementation of the spin component.

<double precision> spin() Get the spin property of an preset implementation of the spin component class.

<type(varying_string), allocatable, dimension(:) => matches> spinAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spin class that have the desired attributes for the spin property

<integer> spinCount() Return a count of the number of scalar properties in the spin property of an preset implementation of the spin component class.

<double precision> spinGrowthRate() Get the spinGrowthRate property of an preset implementation of the spin component class.

<type(varying_string), allocatable, dimension(:) => matches> spinGrowthRateAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spin class that have the desired attributes for the spinGrowthRate property

<logical> spinGrowthRateIsGettable() Returns true if the spinGrowthRate property is gettable for the spin component class.

<logical> spinGrowthRateIsSettable() Specify whether the spinGrowthRate property of the spin component is settable.

<double precision> spinGrowthRateRateGet() Returns a zero rate for the spinGrowthRate property for the spin component class.

<void> spinGrowthRateSet(<double precision> setValue→) Set the spinGrowthRate property of an preset implementation of the spin component class.

<void> spinInactive() Indicate that the spin property of an preset implementation of the spin component class is inactive for differential equation solving.

<logical> spinIsGettable() Returns true if the spin property is gettable for the spin component class.

<logical> spinIsSettable() Specify whether the spin property of the spin component is settable.

<void> spinRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the spin property of an preset implementation of the spin component class.

<double precision> spinRateGet() Get the rate of change of the spin property of an preset implementation of the spin component class.

<void> spinScale(<double precision> setValue→) Set the absolute scale of the spin property of an preset implementation of the spin component class.

<void> spinSet(<double precision> setValue→) Set the spin property of an preset implementation of the spin component class.

<double precision, dimension(:), allocatable => classDefault> spinVector() Returns the default value for the spinVector property for the spin component class.

<type(varying_string), allocatable, dimension(:) => matches> spinVectorAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spin class that have the desired attributes for the spinVector property

<integer> spinVectorCount() Compute the count of evolvable quantities in the spinVector property of the SpinVitvitska component.

<double precision, dimension(:), allocatable => classDefault> spinVectorGrowthRate() Returns the default value for the spinVectorGrowthRate property for the spin component class.

<type(varying_string), allocatable, dimension(:) => matches> spinVectorGrowthRateAttributeMatch(<logic
[requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→)
Return a text list of component implementations in the spin class that have the desired attributes for the spinVectorGrowthRate property

<logical> spinVectorGrowthRateIsGettable() Returns true if the spinVectorGrowthRate property is gettable for the spin component class.

<logical> spinVectorGrowthRateIsSettable() Specify whether the spinVectorGrowthRate property of the spin component is settable.

<double precision, dimension(:), allocatable => classDefault> spinVectorGrowthRateRateGet()
Returns a zero rate for the spinVectorGrowthRate property for the spin component class.

<void> spinVectorGrowthRateSet(<double precision[:]> value) Set the spinVectorGrowthRate property of the spin component.

<logical> spinVectorIsGettable() Returns true if the spinVector property is gettable for the spin component class.

<logical> spinVectorIsSettable() Specify whether the spinVector property of the spin component is settable.

<void> spinVectorRate(<double precision[:]> value) Cumulate to the rate of the spinVector property of the SpinVitvitska component.

<double precision, dimension(:), allocatable => classDefault> spinVectorRateGet() Returns a zero rate for the spinVector property for the spin component class.

<void> spinVectorScale(<double precision[:]> value) Set the scale of the spinVector property of the SpinVitvitska component.

<void> spinVectorSet(<double precision[:]> value) Set the spinVector property of the spin component.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→,
<massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<type(varying_string)> type() Returns the type name for the preset implementation of the spin component class.

<logical> vitvitskaIsActive() Return true if the vitvitska implementation of the spin component is the active choice.

nodeComponentSpinPreset3D

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a preset3D implementation of the spin component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a preset3D implementation of the spin component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a preset3D implementation of the spin component from array.

<void> destroy() Finalize a preset3D implementation of the spin component.

<void> dumpASCII() Dump the content of a spin component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a preset3D member of the spin component.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a preset3D implementation of the spin component class.

<logical> nullIsActive() Return true if the null implementation of the spin component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a spin component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a preset3D implementation of the spin component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a preset3D implementation of the spin component.

<void> postOutput(<double precision> time→) Perform post-output processing for a preset3D implementation of the spin component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<logical> preset3DIsActive() Return true if the preset3D implementation of the spin component is the active choice.

<logical> presetIsActive() Return true if the preset implementation of the spin component is the active choice.

<logical> randomIsActive() Return true if the random implementation of the spin component is the active choice.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a preset3D implementation of the spin component.

<void> serializeASCII() Serialize the contents of a preset3D implementation of the spin component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a preset3D implementation of the spin component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a preset3D implementation of the spin component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a preset3D implementation of the spin component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a preset3D implementation of the spin component to XML.

<integer(c_size_t)> sizeOf() Return the size in bytes of a preset3D implementation of the spin component.

<double precision> spin() Get the spin property of an preset implementation of the spin component class.

<type(varying_string), allocatable, dimension(:) => matches> spinAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spin class that have the desired attributes for the spin property

<integer> spinCount() Return a count of the number of scalar properties in the spin property of an preset implementation of the spin component class.

<double precision> spinGrowthRate() Get the spinGrowthRate property of an preset implementation of the spin component class.

<type(varying_string), allocatable, dimension(:) => matches> spinGrowthRateAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spin class that have the desired attributes for the spinGrowthRate property

<logical> spinGrowthRateIsGettable() Returns true if the spinGrowthRate property is gettable for the spin component class.

<logical> spinGrowthRateIsSettable() Specify whether the spinGrowthRate property of the spin component is settable.

<double precision> spinGrowthRateRateGet() Returns a zero rate for the spinGrowthRate property for the spin component class.

<void> spinGrowthRateSet(<double precision> setValue→) Set the spinGrowthRate property of an preset implementation of the spin component class.

<void> spinInactive() Indicate that the spin property of an preset implementation of the spin component class is inactive for differential equation solving.

<logical> spinIsGettable() Returns true if the spin property is gettable for the spin component class.

<logical> spinIsSettable() Specify whether the spin property of the spin component is settable.

<void> spinRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the spin property of an preset implementation of the spin component class.

<double precision> spinRateGet() Get the rate of change of the spin property of an preset implementation of the spin component class.

<void> spinScale(<double precision> setValue→) Set the absolute scale of the spin property of an preset implementation of the spin component class.

<void> spinSet(<double precision> setValue→) Set the spin property of an preset implementation of the spin component class.

<double precision, dimension(:), allocatable => propertyValue> spinVector() Get the spinVector property of an preset3D implementation of the spin component class.

<type(varying_string), allocatable, dimension(:) => matches> spinVectorAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spin class that have the desired attributes for the spinVector property

<integer> spinVectorCount() Return a count of the number of scalar properties in the spinVector property of an preset3D implementation of the spin component class.

<double precision, dimension(:), allocatable => propertyValue> spinVectorGrowthRate() Get the spinVectorGrowthRate property of an preset3D implementation of the spin component class.

<type(varying_string), allocatable, dimension(:) => matches> spinVectorGrowthRateAttributeMatch(<logic [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spin class that have the desired attributes for the spinVectorGrowthRate property

<logical> spinVectorGrowthRateIsGettable() Returns true if the spinVectorGrowthRate property is gettable for the spin component class.

<logical> spinVectorGrowthRateIsSettable() Specify whether the spinVectorGrowthRate property of the spin component is settable.

<double precision, dimension(:), allocatable => classDefault> spinVectorGrowthRateRateGet() Returns a zero rate for the spinVectorGrowthRate property for the spin component class.

<void> spinVectorGrowthRateSet(<double precision[:]> setValue→) Set the spinVectorGrowthRate property of an preset3D implementation of the spin component class.

<void> spinVectorInactive() Indicate that the spinVector property of an preset3D implementation of the spin component class is inactive for differential equation solving.

<logical> spinVectorIsGettable() Returns true if the spinVector property is gettable for the spin component class.

<logical> spinVectorIsSettable() Specify whether the spinVector property of the spin component is settable.

<void> spinVectorRate(<double precision[:]> setValue→, <logical> [interrupt]↔, <*procedure(interruptT [interruptProcedure]↔) Accumulate to the rate of change of the spinVector property of an preset3D implementation of the spin component class.

<double precision, dimension(:), allocatable => propertyRate> spinVectorRateGet() Get the rate of change of the spinVector property of an preset3D implementation of the spin component class.

<void> spinVectorScale(<double precision[:]> setValue→) Set the absolute scale of the spinVector property of an preset3D implementation of the spin component class.

<void> spinVectorSet(<double precision[:]> setValue→) Set the spinVector property of an preset3D implementation of the spin component class.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<type(varying_string)> type() Returns the type name for the preset3D implementation of the spin component class.

<logical> vitvitskaIsActive() Return true if the vitvitska implementation of the spin component is the active choice.

nodeComponentSpinRandom

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a random implementation of the spin component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a random implementation of the spin component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a random implementation of the spin component from array.

<void> destroy() Finalize a random implementation of the spin component.

<void> dumpASCII() Dump the content of a spin component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a random member of the spin component.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a random implementation of the spin component class.

<logical> nullIsActive() Return true if the null implementation of the spin component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a spin component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a random implementation of the spin component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a random implementation of the spin component.

<void> postOutput(<double precision> time→) Perform post-output processing of a spin component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<logical> preset3DIsActive() Return true if the preset3D implementation of the spin component is the active choice.

<logical> presetIsActive() Return true if the preset implementation of the spin component is the active choice.

<logical> randomIsActive() Return true if the random implementation of the spin component is the active choice.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a random implementation of the spin component.

<void> serializeASCII() Serialize the contents of a random implementation of the spin component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a random implementation of the spin component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a random implementation of the spin component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a random implementation of the spin component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a random implementation of the spin component to XML.

<integer(c_size_t)> sizeOf() Return the size in bytes of a random implementation of the spin component.

<double precision> spin() Get the spin property of an random implementation of the spin component class.

<type(varying_string), allocatable, dimension(:) => matches> spinAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spin class that have the desired attributes for the spin property

<integer> spinCount() Return a count of the number of scalar properties in the spin property of an random implementation of the spin component class.

<double precision> spinGrowthRate() Returns the default value for the spinGrowthRate property for the spin component class.

<type(varying_string), allocatable, dimension(:) => matches> spinGrowthRateAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spin class that have the desired attributes for the spinGrowthRate property

<logical> spinGrowthRateIsGettable() Returns true if the spinGrowthRate property is gettable for the spin component class.

<logical> spinGrowthRateIsSettable() Specify whether the spinGrowthRate property of the spin component is settable.

<double precision> spinGrowthRateRateGet() Returns a zero rate for the spinGrowthRate property for the spin component class.

<void> spinGrowthRateSet(<double precision> value) Set the spinGrowthRate property of the spin component.

<void> spinInactive() Indicate that the spin property of an random implementation of the spin component class is inactive for differential equation solving.

<logical> spinIsGettable() Returns true if the spin property is gettable for the spin component class.

<logical> spinIsSettable() Specify whether the spin property of the spin component is settable.

<void> spinRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the spin property of an random implementation of the spin component class.

<double precision> spinRateGet() Get the rate of change of the spin property of an random implementation of the spin component class.

<void> spinScale(<double precision> setValue→) Set the absolute scale of the spin property of an random implementation of the spin component class.

<void> spinSet(<double precision> setValue→) Set the spin property of an random implementation of the spin component class.

<double precision, dimension(:), allocatable => classDefault> spinVector() Returns the default value for the spinVector property for the spin component class.

<type(varying_string), allocatable, dimension(:) => matches> spinVectorAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spin class that have the desired attributes for the spinVector property

<integer> spinVectorCount() Compute the count of evolvable quantities in the spinVector property of the SpinVitvitska component.

<double precision, dimension(:), allocatable => classDefault> spinVectorGrowthRate() Returns the default value for the spinVectorGrowthRate property for the spin component class.

<type(varying_string), allocatable, dimension(:) => matches> spinVectorGrowthRateAttributeMatch(<logic [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spin class that have the desired attributes for the spinVectorGrowthRate property

<logical> spinVectorGrowthRateIsGettable() Returns true if the spinVectorGrowthRate property is gettable for the spin component class.

<logical> spinVectorGrowthRateIsSettable() Specify whether the spinVectorGrowthRate property of the spin component is settable.

<double precision, dimension(:), allocatable => classDefault> spinVectorGrowthRateRateGet() Returns a zero rate for the spinVectorGrowthRate property for the spin component class.

<void> spinVectorGrowthRateSet(<double precision[:]> value) Set the spinVectorGrowthRate property of the spin component.

<logical> spinVectorIsGettable() Returns true if the spinVector property is gettable for the spin component class.

<logical> spinVectorIsSettable() Specify whether the spinVector property of the spin component is settable.

<void> spinVectorRate(<double precision[:]> value) Cumulate to the rate of the spinVector property of the SpinVitvitska component.

<double precision, dimension(:), allocatable => classDefault> spinVectorRateGet() Returns a zero rate for the spinVector property for the spin component class.

<void> spinVectorScale(<double precision[:]> value) Set the scale of the spinVector property of the SpinVitvitska component.

<void> spinVectorSet(<double precision[:]> value) Set the spinVector property of the spin component.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<type(varying_string)> type() Returns the type name for the random implementation of the spin component class.

<logical> vitvitskaIsActive() Return true if the vitvitska implementation of the spin component is the active choice.

nodeComponentSpinVitvitska

<void> assign(<class(nodeComponent)> to←, <class(nodeComponent)> from→) Assign a node component to another node component.

<void> builder(<*type(node)> componentDefinition→) Build a vitvitska implementation of the spin component from a supplied XML definition.

<double> density(<double(3)> positionSpherical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the density.

<void> deserializeRaw(<integer> fileHandle→) Deserialize the contents of a vitvitska implementation of the spin component from raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a vitvitska implementation of the spin component from array.

<void> destroy() Finalize a vitvitska implementation of the spin component.

<void> dumpASCII() Dump the content of a spin component.

<double> enclosedMass(<double> radius→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the mass enclosed within a radius.

<*type(treeNode)> host() Return a pointer to the host treeNode object.

<void> hotHaloCoolingAbundancesRate(<type(abundances)> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingAngularMomentumRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.

<void> hotHaloCoolingMassRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(inter [interruptProcedure]↔) Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.

<void> initialize() Initialize a vitvitska member of the spin component.

<type(varying_string)> nameFromIndex(<integer> count↔, <integer> propertyType→) Return the name of the property of given index for a vitvitska implementation of the spin component class.

<logical> nullIsActive() Return true if the null implementation of the spin component is the active choice.

<void> odeStepRatesInitialize() Initialize rates for evolvable properties.

<void> odeStepScalesInitialize() Initialize scales for evolvable properties.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→, <integer> instance→) Populate output buffers with properties to output for a spin component.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→, <integer> instance→) Increment the count of properties to output for a vitvitska implementation of the spin component.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→, <integer> instance→) Return the names of properties to output for a vitvitska implementation of the spin component.

<void> postOutput(<double precision> time→) Perform post-output processing of a spin component.

<double> potential(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the gravitational potential.

<logical> preset3DIsActive() Return true if the preset3D implementation of the spin component is the active choice.

<logical> presetIsActive() Return true if the preset implementation of the spin component is the active choice.

<logical> randomIsActive() Return true if the random implementation of the spin component is the active choice.

<double> rotationCurve(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve.

<double> rotationCurveGradient(<double> radius→, <componentType> [componentType]→, <massType> [massType]→) Compute the rotation curve gradient.

<void> serializationOffsets(<integer> count↔, <integer> countSubset↔, <integer> propertyType→) Compute offsets into serialization arrays for a vitvitska implementation of the spin component.

<void> serializeASCII() Serialize the contents of a vitvitska implementation of the spin component to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the serialization of a vitvitska implementation of the spin component.

<void> serializeRaw(<integer> fileHandle→) Serialize the contents of a vitvitska implementation of the spin component to raw (binary) file.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a vitvitska implementation of the spin component to array.

<void> serializeXML(<integer> fileHandle→) Serialize the contents of a vitvitska implementation of the spin component to XML.

<integer(c_size_t)> sizeOf() Return the size in bytes of a vitvitska implementation of the spin component.

<double precision> spin() Get the value of the spin property of the vitvitska implementation of the spin component using a deferred function.

<type(varying_string), allocatable, dimension(:) => matches> spinAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spin class that have the desired attributes for the spin property

<integer> spinCount() Return a count of the number of scalar properties in the spin property of an vitvitska implementation of the spin component class.

<void> spinFunction() Set the function to be used for the get method of the spin property of the SpinVitvitska component.

<double precision> spinGrowthRate() Get the value of the spinGrowthRate property of the vitvitska implementation of the spin component using a deferred function.

<type(varying_string), allocatable, dimension(:) => matches> spinGrowthRateAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spin class that have the desired attributes for the spinGrowthRate property

<void> spinGrowthRateFunction() Set the function to be used for the get method of the spinGrowthRate property of the SpinVitvitska component.

<logical> spinGrowthRateIsAttached() Return true if the deferred function used to get the spinGrowthRate property of the SpinVitvitska component class has been attached.

<logical> spinGrowthRateIsGettable() Returns true if the spinGrowthRate property is gettable for the spin component class.

<logical> spinGrowthRateIsSettable() Specify whether the spinGrowthRate property of the spin component is settable.

<double precision> spinGrowthRateRateGet() Returns a zero rate for the spinGrowthRate property for the spin component class.

<void> spinGrowthRateSet(<double precision> value) Set the spinGrowthRate property of the spin component.

<void> spinInactive() Indicate that the spin property of an vitvitska implementation of the spin component class is inactive for differential equation solving.

<logical> spinIsAttached() Return true if the deferred function used to get the spin property of the SpinVitvitska component class has been attached.

<logical> spinIsGettable() Returns true if the spin property is gettable for the spin component class.

<logical> spinIsSettable() Specify whether the spin property of the spin component is settable.

<void> spinRate(<double precision> setValue→, <logical> [interrupt]↔, <*procedure(interruptTask)> [interruptProcedure]↔) Accumulate to the rate of change of the spin property of an vitvitska implementation of the spin component class.

<double precision> spinRateGet() Get the rate of change of the spin property of an vitvitska implementation of the spin component class.

<void> spinScale(<double precision> setValue→) Set the absolute scale of the spin property of an vitvitska implementation of the spin component class.

<void> spinSet(<double precision> setValue→) Set the spin property of an vitvitska implementation of the spin component class.

<double precision> spinValue() Get the spin property of an vitvitska implementation of the spin component class.

<double precision, dimension(:), allocatable => propertyValue> spinVector() Get the value of the spinVector property of the vitvitska implementation of the spin component using a deferred function.

<type(varying_string), allocatable, dimension(:) => matches> spinVectorAttributeMatch(<logical> [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spin class that have the desired attributes for the spinVector property

<integer> spinVectorCount() Return a count of the number of scalar properties in the spinVector property of an vitvitska implementation of the spin component class.

<void> spinVectorFunction() Set the function to be used for the get method of the spinVector property of the SpinVitvitska component.

<double precision, dimension(:), allocatable => classDefault> spinVectorGrowthRate() Returns the default value for the spinVectorGrowthRate property for the spin component class.

<type(varying_string), allocatable, dimension(:) => matches> spinVectorGrowthRateAttributeMatch(<logic [requireSettable]→, <logical> [requireGettable]→, <logical> [requireEvolvable]→) Return a text list of component implementations in the spin class that have the desired attributes for the spinVectorGrowthRate property

<logical> spinVectorGrowthRateIsGettable() Returns true if the spinVectorGrowthRate property is gettable for the spin component class.

<logical> spinVectorGrowthRateIsSettable() Specify whether the spinVectorGrowthRate property of the spin component is settable.

<double precision, dimension(:), allocatable => classDefault> spinVectorGrowthRateRateGet() Returns a zero rate for the spinVectorGrowthRate property for the spin component class.

<void> spinVectorGrowthRateSet(<double precision[:]> value) Set the spinVectorGrowthRate property of the spin component.

<void> spinVectorInactive() Indicate that the spinVector property of an vitvitska implementation of the spin component class is inactive for differential equation solving.

<logical> spinVectorIsAttached() Return true if the deferred function used to get the spinVector property of the SpinVitvitska component class has been attached.

<logical> spinVectorIsGettable() Returns true if the spinVector property is gettable for the spin component class.

<logical> spinVectorIsSettable() Specify whether the spinVector property of the spin component is settable.

<void> spinVectorRate(<double precision[:]> setValue→, <logical> [interrupt]↔, <*procedure(interruptT [interruptProcedure]↔) Accumulate to the rate of change of the spinVector property of an vitvitska implementation of the spin component class.

<double precision, dimension(:), allocatable => propertyRate> spinVectorRateGet() Get the rate of change of the spinVector property of an vitvitska implementation of the spin component class.

<void> spinVectorScale(<double precision[:]> setValue→) Set the absolute scale of the spinVector property of an vitvitska implementation of the spin component class.

<void> spinVectorSet(<double precision[:]> setValue→) Set the spinVector property of an vitvitska implementation of the spin component class.

<double precision, dimension(:), allocatable => propertyValue> spinVectorValue() Get the spinVector property of an vitvitska implementation of the spin component class.

<double> surfaceDensity(<double(3)> positionCylindrical→, <componentType> [componentType]→, <massType> [massType]→, <weightBy> [weightBy]→, <integer> [weightIndex]→) Compute the surface density.

<type(varying_string)> type() Returns the type name for the vitvitska implementation of the spin component class.

<logical> vitvitskaIsActive() Return true if the vitvitska implementation of the spin component is the active choice.

nodeEvent

<void> deserializeRaw(<integer> fileUnit→) Deserialize a nodeEvent object from raw file.

<integer(c_size_t)> nonStaticSizeOf() Compute the size of the non-static parts of a spin object.

<void> serializeRaw(<integer> fileUnit→, <logical> includeType→) Serialize a nodeEvent object to raw file.

nodeEventBranchJump

<void> deserializeRaw(<integer> fileUnit→) Deserialize a nodeEventBranchJump object from raw file.

<integer(c_size_t)> nonStaticSizeOf() Compute the size of the non-static parts of a spin object.

<void> serializeRaw(<integer> fileUnit→, <logical> includeType→) Serialize a nodeEventBranchJump object to raw file.

nodeEventBranchJumpInterTree

<void> deserializeRaw(<integer> fileUnit→) Deserialize a nodeEventBranchJumpInterTree object from raw file.

<integer(c_size_t)> nonStaticSizeOf() Compute the size of the non-static parts of a spin object.

<void> serializeRaw(<integer> fileUnit→, <logical> includeType→) Serialize a nodeEventBranchJumpInterTree object to raw file.

nodeEventSubhaloPromotion

<void> deserializeRaw(<integer> fileUnit→) Deserialize a nodeEventSubhaloPromotion object from raw file.

<integer(c_size_t)> nonStaticSizeOf() Compute the size of the non-static parts of a spin object.

<void> serializeRaw(<integer> fileUnit→, <logical> includeType→) Serialize a nodeEventSubhaloPromotion object to raw file.

nodeEventSubhaloPromotionInterTree

<void> deserializeRaw(<integer> fileUnit→) Deserialize a nodeEventSubhaloPromotionInterTree object from raw file.

<integer(c_size_t)> nonStaticSizeOf() Compute the size of the non-static parts of a spin object.

<void> serializeRaw(<integer> fileUnit→, <logical> includeType→) Serialize a nodeEventSubhaloPromotionInt object to raw file.

nodePropertyExtractorClass

void addInstances(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red type((multiCounter))\textgreater} instance\arginout) Add multiple instances of this property to a multiCounter object.

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((nodePropertyExtractorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

integer quantity() Return the class of the extracted property.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

integer type() Return the type of the extracted property.

nodePropertyExtractorConcentration

void addInstances(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red type((multiCounter))\textgreater} instance\arginout) Add multiple instances of this property to a multiCounter object.

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((nodePropertyExtractorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

<type(varying_string)> description() Return a description of the property extracted.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

<double> extract(<type(treeNode)> node→) Extract the property from the given node.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

<type(varying_string)> name() Return the name of the property extracted.

type(varying_string) objectType() Return the type of the object.

integer quantity() Return the class of the extracted property.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

integer type() Return the type of the extracted property.

<double> unitsInSI() Return the units of the property extracted in the SI system.

nodePropertyExtractorDensityContrasts

void addInstances(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red type((multiCounter))\textgreater} instance\arginout) Add multiple instances of this property to a multiCounter object.

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((nodePropertyExtractorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

<type(varying_string)>(:) descriptions(<double> time→) Return descriptions of the properties extracted.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

<integer> elementCount(<double> time→) Return the number of properties in the tuple.

<double(:)> extract(<type(treeNode)> node→, <double> time→, <type(multiCounter)> [instance]→) Extract the properties from the given node.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

<type(varying_string)>(:) names(<double> time→) Return the names of the properties extracted.

type(varying_string) objectType() Return the type of the object.

integer quantity() Return the class of the extracted property.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

integer type() Return the type of the extracted property.

<double(:)> unitsInSI(<double> time→) Return the units of the properties extracted in the SI system.

nodePropertyExtractorDensityProfile

void addInstances(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red type((multiCounter))\textgreater} instance\arginout) Add multiple instances of this property to a multiCounter object.

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((nodePropertyExtractorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

<type(varying_string)>(:) descriptions(<double> time→) Return descriptions of the properties extracted.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

<integer> elementCount(<double> time→) Return the number of properties in the tuple.

<double(:)> extract(<type(treeNode)> node→, <double> time→, <type(multiCounter)> [instance]→) Extract the properties from the given node.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

<type(varying_string)>(:) names(<double> time→) Return the names of the properties extracted.

type(varying_string) objectType() Return the type of the object.

integer quantity() Return the class of the extracted property.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

integer type() Return the type of the extracted property.

<double(:)> unitsInSI(<double> time→) Return the units of the properties extracted in the SI system.

nodePropertyExtractorDescendents

void addInstances(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red type((multiCounter))\textgreater} instance\arginout) Add multiple instances of this property to a multiCounter object.

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((nodePropertyExtractorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

<type(varying_string)> description() Return a description of the property extracted.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

<integer> extract(<type(treeNode)> node→) Extract the property from the given node.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

<type(varying_string)> name() Return the name of the property extracted.

type(varying_string) objectType() Return the type of the object.

integer quantity() Return the class of the extracted property.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

integer type() Return the type of the extracted property.

<double> unitsInSI() Return the units of the property extracted in the SI system.

nodePropertyExtractorFinalDescendent

void addInstances(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless type((multiCounter))\textgreater} instance\arginout) Add multiple instances of this property to a multiCounter object.

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\arginout,\textcolor{red}{\textless character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((nodePropertyExtractorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

<type(varying_string)> description() Return a description of the property extracted.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\textcolor{red}{\textless logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

<integer> extract(<type(treeNode)> node→) Extract the property from the given node.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDigest\argin) Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

<type(varying_string)> name() Return the name of the property extracted.

type(varying_string) objectType() Return the type of the object.

integer quantity() Return the class of the extracted property.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`integer type()` Return the type of the extracted property.

`<double> unitsInSI()` Return the units of the property extracted in the SI system.

nodePropertyExtractorFractionAccretionHotMode

`void addInstances(\textcolor{red}{\textless type((treeNode ))\textgreater} node\arginout,\textcolor{red type((multiCounter))\textgreater} instance\arginout)` Add multiple instances of this property to a `multiCounter` object.

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((nodePropertyExtractorClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`<type(varying_string)> description()` Return a description of the property extracted.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`<double> extract(<type(treeNode)> node→)` Extract the property from the given `node`.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`<type(varying_string)> name()` Return the name of the property extracted.

`type(varying_string) objectType()` Return the type of the object.

`integer quantity()` Return the class of the extracted property.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`integer type()` Return the type of the extracted property.

`<double> unitsInSI()` Return the units of the property extracted in the SI system.

nodePropertyExtractorHalfMassRadius

void addInstances(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless type((multiCounter))\textgreater} instance\arginout) Add multiple instances of this property to a multiCounter object.

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\arginout,\textcolor{red}{\textless character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((nodePropertyExtractorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

<type(varying_string)> description() Return a description of the property extracted.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\textcolor{red}{\textless logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

<double> extract(<type(treeNode)> node→) Extract the property from the given node.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDigest\argin) Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

<type(varying_string)> name() Return the name of the property extracted.

type(varying_string) objectType() Return the type of the object.

integer quantity() Return the class of the extracted property.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

integer type() Return the type of the extracted property.

<double> unitsInSI() Return the units of the property extracted in the SI system.

nodePropertyExtractorHaloBias

`void addInstances(\textcolor{red}{\textless type((treeNode ))\textgreater} node\arginout,\textcolor{red}{\textless type((multiCounter))\textgreater} instance\arginout)` Add multiple instances of this property to a `multiCounter` object.

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\arginout,\textcolor{red}{\textless character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((nodePropertyExtractorClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`<type(varying_string)> description()` Return a description of the property extracted.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\textcolor{red}{\textless logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`<double> extract(<type(treeNode)> node→)` Extract the property from the given `node`.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDigest\argin)` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`<type(varying_string)> name()` Return the name of the property extracted.

`type(varying_string) objectType()` Return the type of the object.

`integer quantity()` Return the class of the extracted property.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`integer type()` Return the type of the extracted property.

`<double> unitsInSI()` Return the units of the property extracted in the SI system.

nodePropertyExtractorHaloEnvironment

void addInstances(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red} type((multiCounter))\textgreater} instance\arginout) Add multiple instances of this property to a multiCounter object.

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((nodePropertyExtractorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

<type(varying_string)>(:) descriptions(<double> time→) Return descriptions of the properties extracted.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

<integer> elementCount(<double> time→) Return the number of properties in the tuple.

<double(:)> extract(<type(treeNode)> node→, <double> time→, <type(multiCounter)> [instance]→) Extract the properties from the given node.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

<type(varying_string)>(:) names(<double> time→) Return the names of the properties extracted.

type(varying_string) objectType() Return the type of the object.

integer quantity() Return the class of the extracted property.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

integer type() Return the type of the extracted property.

<double(:)> unitsInSI(<double> time→) Return the units of the properties extracted in the SI system.

nodePropertyExtractorICMSZ

`void addInstances(\textcolor{red}{\textless type((treeNode ))\textgreater} node\arginout,\textcolor{red}{\textless type((multiCounter))\textgreater} instance\arginout)` Add multiple instances of this property to a `multiCounter` object.

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\arginout,\textcolor{red}{\textless character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((nodePropertyExtractorClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`<type(varying_string)> description()` Return a description of the property extracted.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\textcolor{red}{\textless logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`<double> extract(<type(treeNode)> node→)` Extract the property from the given `node`.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDigest\argin)` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`<type(varying_string)> name()` Return the name of the property extracted.

`type(varying_string) objectType()` Return the type of the object.

`integer quantity()` Return the class of the extracted property.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`integer type()` Return the type of the extracted property.

`<double> unitsInSI()` Return the units of the property extracted in the SI system.

nodePropertyExtractorICMXRayLuminosity

`void addInstances(\textcolor{red}{\textless type((treeNode ))\textgreater} node\arginout,\textcolor{red}{\textless type((multiCounter))\textgreater} instance\arginout)` Add multiple instances of this property to a `multiCounter` object.

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\arginout,\textcolor{red}{\textless character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((nodePropertyExtractorClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`<type(varying_string)>(:) descriptions(<double> time→)` Return descriptions of the properties extracted.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\textcolor{red}{\textless logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`<integer> elementCount(<double> time→)` Return the number of properties in the tuple.

`<double(:)> extract(<type(treeNode)> node→, <double> time→, <type(multiCounter)> [instance]→)` Extract the properties from the given `node`.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDigest\argin)` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`<type(varying_string)>(:) names(<double> time→)` Return the names of the properties extracted.

`type(varying_string) objectType()` Return the type of the object.

`integer quantity()` Return the class of the extracted property.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`integer type()` Return the type of the extracted property.

`<double(:)> unitsInSI(<double> time→)` Return the units of the properties extracted in the SI system.

nodePropertyExtractorIndicesHost

`void addInstances(\textcolor{red}{\textless type((treeNode ))\textgreater} node\arginout,\textcolor{red}{\textless type((multiCounter))\textgreater} instance\arginout)` Add multiple instances of this property to a `multiCounter` object.

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\arginout,\textcolor{red}{\textless character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((nodePropertyExtractorClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`<type(varying_string)> description()` Return a description of the property extracted.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\textcolor{red}{\textless logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`<integer> extract(<type(treeNode)> node→)` Extract the property from the given `node`.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDigest\argin)` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`<type(varying_string)> name()` Return the name of the property extracted.

`type(varying_string) objectType()` Return the type of the object.

`integer quantity()` Return the class of the extracted property.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`integer type()` Return the type of the extracted property.

`<double> unitsInSI()` Return the units of the property extracted in the SI system.

nodePropertyExtractorIndicesTree

void addInstances(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red} type((multiCounter))\textgreater} instance\arginout) Add multiple instances of this property to a multiCounter object.

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((nodePropertyExtractorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

<type(varying_string)> description() Return a description of the property extracted.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

<integer> extract(<type(treeNode)> node→) Extract the property from the given node.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

<type(varying_string)> name() Return the name of the property extracted.

type(varying_string) objectType() Return the type of the object.

integer quantity() Return the class of the extracted property.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

integer type() Return the type of the extracted property.

<double> unitsInSI() Return the units of the property extracted in the SI system.

nodePropertyExtractorIntegerScalar

`void addInstances(\textcolor{red}{\textless type((treeNode ))\textgreater} node\arginout,\textcolor{red}{\textless type((multiCounter))\textgreater} instance\arginout)` Add multiple instances of this property to a `multiCounter` object.

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\arginout,\textcolor{red}{\textless character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((nodePropertyExtractorClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`<type(varying_string)> description()` Return a description of the property extracted.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\textcolor{red}{\textless logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`<integer> extract(<type(treeNode)> node→)` Extract the property from the given `node`.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDigest\argin)` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`<type(varying_string)> name()` Return the name of the property extracted.

`type(varying_string) objectType()` Return the type of the object.

`integer quantity()` Return the class of the extracted property.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`integer type()` Return the type of the extracted property.

`<double> unitsInSI()` Return the units of the property extracted in the SI system.

nodePropertyExtractorIntegerTuple

void addInstances(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red} type((multiCounter))\textgreater} instance\arginout) Add multiple instances of this property to a multiCounter object.

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((nodePropertyExtractorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

<type(varying_string)>(:) descriptions(<double> time→) Return descriptions of the properties extracted.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

<integer> elementCount(<double> time→) Return the number of properties in the tuple.

<integer(:)> extract(<type(treeNode)> node→, <double> time→, <type(multiCounter)> [instance]→) Extract the properties from the given node.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

<type(varying_string)>(:) names(<double> time→) Return the names of the properties extracted.

type(varying_string) objectType() Return the type of the object.

integer quantity() Return the class of the extracted property.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

integer type() Return the type of the extracted property.

<double(:)> unitsInSI(<double> time→) Return the units of the properties extracted in the SI system.

nodePropertyExtractorLightcone

`void addInstances(\textcolor{red}{\textless type((treeNode ))\textgreater} node\arginout,\textcolor{red}{\textless type((multiCounter))\textgreater} instance\arginout)` Add multiple instances of this property to a `multiCounter` object.

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\arginout,\textcolor{red}{\textless character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((nodePropertyExtractorClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`<type(varying_string)>(:) descriptions(<double> time→)` Return descriptions of the properties extracted.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\textcolor{red}{\textless logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`<integer> elementCount(<double> time→)` Return the number of properties in the tuple.

`<double(:)> extract(<type(treeNode)> node→, <double> time→, <type(multiCounter)> [instance]→)` Extract the properties from the given `node`.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDigest\argin)` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`<type(varying_string)>(:) names(<double> time→)` Return the names of the properties extracted.

`type(varying_string) objectType()` Return the type of the object.

`integer quantity()` Return the class of the extracted property.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`integer type()` Return the type of the extracted property.

`<double(:)> unitsInSI(<double> time→)` Return the units of the properties extracted in the SI system.

nodePropertyExtractorLmnstyEmssnLine

void addInstances(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red} type((multiCounter))\textgreater} instance\arginout) Add multiple instances of this property to a multiCounter object.

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((nodePropertyExtractorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

<type(varying_string)> description() Return a description of the property extracted.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

<double> extract(<type(treeNode)> node→) Extract the property from the given node.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

<type(varying_string)> name() Return the name of the property extracted.

type(varying_string) objectType() Return the type of the object.

integer quantity() Return the class of the extracted property.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

integer type() Return the type of the extracted property.

<double> unitsInSI() Return the units of the property extracted in the SI system.

nodePropertyExtractorLmnstyStllrCF2000

`void addInstances(\textcolor{red}{\textless type((treeNode ))\textgreater} node\arginout,\textcolor{red}{\textless type((multiCounter))\textgreater} instance\arginout)` Add multiple instances of this property to a `multiCounter` object.

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\arginout,\textcolor{red}{\textless character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((nodePropertyExtractorClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`<type(varying_string)> description()` Return a description of the property extracted.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\textcolor{red}{\textless logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`<double> extract(<type(treeNode)> node→)` Extract the property from the given `node`.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDigest\argin)` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`<type(varying_string)> name()` Return the name of the property extracted.

`type(varying_string) objectType()` Return the type of the object.

`integer quantity()` Return the class of the extracted property.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`integer type()` Return the type of the extracted property.

`<double> unitsInSI()` Return the units of the property extracted in the SI system.

nodePropertyExtractorLuminosityStellar

void addInstances(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless type((multiCounter))\textgreater} instance\arginout) Add multiple instances of this property to a multiCounter object.

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\arginout,\textcolor{red}{\textless character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((nodePropertyExtractorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

<type(varying_string)> description() Return a description of the property extracted.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\textcolor{red}{\textless logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

<double> extract(<type(treeNode)> node→) Extract the property from the given node.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDigest\argin) Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

<type(varying_string)> name() Return the name of the property extracted.

type(varying_string) objectType() Return the type of the object.

integer quantity() Return the class of the extracted property.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

integer type() Return the type of the extracted property.

<double> unitsInSI() Return the units of the property extracted in the SI system.

nodePropertyExtractorMainBranchStatus

`void addInstances(\textcolor{red}{\textless type((treeNode ))\textgreater} node\arginout,\textcolor{red}{\textless type((multiCounter))\textgreater} instance\arginout)` Add multiple instances of this property to a `multiCounter` object.

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\arginout,\textcolor{red}{\textless character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((nodePropertyExtractorClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`<type(varying_string)> description()` Return a description of the property extracted.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\textcolor{red}{\textless logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`<integer> extract(<type(treeNode)> node→)` Extract the property from the given `node`.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDigest\argin)` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`<type(varying_string)> name()` Return the name of the property extracted.

`type(varying_string) objectType()` Return the type of the object.

`integer quantity()` Return the class of the extracted property.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`integer type()` Return the type of the extracted property.

`<double> unitsInSI()` Return the units of the property extracted in the SI system.

nodePropertyExtractorMassBlackHole

void addInstances(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red} type((multiCounter))\textgreater} instance\arginout) Add multiple instances of this property to a multiCounter object.

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((nodePropertyExtractorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

<type(varying_string)> description() Return a description of the property extracted.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

<double> extract(<type(treeNode)> node→) Extract the property from the given node.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

<type(varying_string)> name() Return the name of the property extracted.

type(varying_string) objectType() Return the type of the object.

integer quantity() Return the class of the extracted property.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

integer type() Return the type of the extracted property.

<double> unitsInSI() Return the units of the property extracted in the SI system.

nodePropertyExtractorMassHalo

`void addInstances(\textcolor{red}{\textless type((treeNode ))\textgreater} node\arginout,\textcolor{red}{\textless type((multiCounter))\textgreater} instance\arginout)` Add multiple instances of this property to a `multiCounter` object.

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\arginout,\textcolor{red}{\textless character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((nodePropertyExtractorClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`<type(varying_string)> description()` Return a description of the property extracted.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\textcolor{red}{\textless logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`<double> extract(<type(treeNode)> node→)` Extract the property from the given `node`.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDigest\argin)` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`<type(varying_string)> name()` Return the name of the property extracted.

`type(varying_string) objectType()` Return the type of the object.

`integer quantity()` Return the class of the extracted property.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`integer type()` Return the type of the extracted property.

`<double> unitsInSI()` Return the units of the property extracted in the SI system.

nodePropertyExtractorMassHost

void addInstances(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red} type((multiCounter))\textgreater} instance\arginout) Add multiple instances of this property to a multiCounter object.

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((nodePropertyExtractorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

<type(varying_string)> description() Return a description of the property extracted.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

<double> extract(<type(treeNode)> node→) Extract the property from the given node.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

<type(varying_string)> name() Return the name of the property extracted.

type(varying_string) objectType() Return the type of the object.

integer quantity() Return the class of the extracted property.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

integer type() Return the type of the extracted property.

<double> unitsInSI() Return the units of the property extracted in the SI system.

nodePropertyExtractorMassISM

`void addInstances(\textcolor{red}{\textless type((treeNode ))\textgreater} node\arginout,\textcolor{red}{\textless type((multiCounter))\textgreater} instance\arginout)` Add multiple instances of this property to a `multiCounter` object.

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\arginout,\textcolor{red}{\textless character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((nodePropertyExtractorClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`<type(varying_string)> description()` Return a description of the property extracted.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\textcolor{red}{\textless logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`<double> extract(<type(treeNode)> node→)` Extract the property from the given `node`.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDigest\argin)` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`<type(varying_string)> name()` Return the name of the property extracted.

`type(varying_string) objectType()` Return the type of the object.

`integer quantity()` Return the class of the extracted property.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`integer type()` Return the type of the extracted property.

`<double> unitsInSI()` Return the units of the property extracted in the SI system.

nodePropertyExtractorMassProfile

void addInstances(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red} type((multiCounter))\textgreater} instance\arginout) Add multiple instances of this property to a multiCounter object.

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((nodePropertyExtractorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

<type(varying_string)>(:) descriptions(<double> time→) Return descriptions of the properties extracted.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

<integer> elementCount(<double> time→) Return the number of properties in the tuple.

<double(:)> extract(<type(treeNode)> node→, <double> time→, <type(multiCounter)> [instance]→) Extract the properties from the given node.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

<type(varying_string)>(:) names(<double> time→) Return the names of the properties extracted.

type(varying_string) objectType() Return the type of the object.

integer quantity() Return the class of the extracted property.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

integer type() Return the type of the extracted property.

<double(:)> unitsInSI(<double> time→) Return the units of the properties extracted in the SI system.

nodePropertyExtractorMassStellar

`void addInstances(\textcolor{red}{\textless type((treeNode ))\textgreater} node\arginout,\textcolor{red}{\textless type((multiCounter))\textgreater} instance\arginout)` Add multiple instances of this property to a `multiCounter` object.

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\arginout,\textcolor{red}{\textless character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((nodePropertyExtractorClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`<type(varying_string)> description()` Return a description of the property extracted.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\textcolor{red}{\textless logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`<double> extract(<type(treeNode)> node→)` Extract the property from the given `node`.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDigest\argin)` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`<type(varying_string)> name()` Return the name of the property extracted.

`type(varying_string) objectType()` Return the type of the object.

`integer quantity()` Return the class of the extracted property.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`integer type()` Return the type of the extracted property.

`<double> unitsInSI()` Return the units of the property extracted in the SI system.

nodePropertyExtractorMassStellarMorphology

void addInstances(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless type((multiCounter))\textgreater} instance\arginout) Add multiple instances of this property to a multiCounter object.

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\arginout,\textcolor{red}{\textless character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((nodePropertyExtractorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

<type(varying_string)> description() Return a description of the property extracted.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\textcolor{red}{\textless logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

<double> extract(<type(treeNode)> node→) Extract the property from the given node.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDigest\argin) Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

<type(varying_string)> name() Return the name of the property extracted.

type(varying_string) objectType() Return the type of the object.

integer quantity() Return the class of the extracted property.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

integer type() Return the type of the extracted property.

<double> unitsInSI() Return the units of the property extracted in the SI system.

nodePropertyExtractorMassStellarSpheroid

`void addInstances(\textcolor{red}{\textless type((treeNode ))\textgreater} node\arginout,\textcolor{red}{\textless type((multiCounter))\textgreater} instance\arginout)` Add multiple instances of this property to a `multiCounter` object.

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\arginout,\textcolor{red}{\textless character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((nodePropertyExtractorClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`<type(varying_string)> description()` Return a description of the property extracted.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\textcolor{red}{\textless logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`<double> extract(<type(treeNode)> node→)` Extract the property from the given `node`.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDigest\argin)` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`<type(varying_string)> name()` Return the name of the property extracted.

`type(varying_string) objectType()` Return the type of the object.

`integer quantity()` Return the class of the extracted property.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`integer type()` Return the type of the extracted property.

`<double> unitsInSI()` Return the units of the property extracted in the SI system.

nodePropertyExtractorMetallicityISM

void addInstances(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red} type((multiCounter))\textgreater} instance\arginout) Add multiple instances of this property to a multiCounter object.

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((nodePropertyExtractorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

<type(varying_string)> description() Return a description of the property extracted.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

<double> extract(<type(treeNode)> node→) Extract the property from the given node.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

<type(varying_string)> name() Return the name of the property extracted.

type(varying_string) objectType() Return the type of the object.

integer quantity() Return the class of the extracted property.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

integer type() Return the type of the extracted property.

<double> unitsInSI() Return the units of the property extracted in the SI system.

nodePropertyExtractorMostMassiveProgenitor

`void addInstances(\textcolor{red}{\textless type((treeNode ))\textgreater} node\arginout,\textcolor{red}{\textless type((multiCounter))\textgreater} instance\arginout)` Add multiple instances of this property to a `multiCounter` object.

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\arginout,\textcolor{red}{\textless character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((nodePropertyExtractorClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`<type(varying_string)> description()` Return a description of the property extracted.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\textcolor{red}{\textless logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`<integer> extract(<type(treeNode)> node→)` Extract the property from the given `node`.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDigest\argin)` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`<type(varying_string)> name()` Return the name of the property extracted.

`type(varying_string) objectType()` Return the type of the object.

`integer quantity()` Return the class of the extracted property.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`integer type()` Return the type of the extracted property.

`<double> unitsInSI()` Return the units of the property extracted in the SI system.

nodePropertyExtractorMulti

`void addInstances(\textcolor{red}{\textless type((treeNode ))\textgreater} node\arginout,\textcolor{red}{\textless type((multiCounter))\textgreater} instance\arginout)` Add multiple instances of this property to a `multiCounter` object.

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\arginout,\textcolor{red}{\textless character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((nodePropertyExtractorClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`<type(varying_string)>(:) descriptions(<integer> elementType→,<double> time→)` Return descriptions of the properties extracted.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\textcolor{red}{\textless logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`<integer> elementCount(<integer> elementType→,<double> time→)` Return the number of properties in the tuple.

`<double(:)> extractDouble(<type(treeNode)> node→, <double> time→, <type(multiCounter)> [instance]→)` Extract the double properties from the given `node`.

`<integer(:)> extractInteger(<type(treeNode)> node→, <double> time→, <type(multiCounter)> [instance]→)` Extract the integer properties from the given `node`.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDigest)` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<void> initialize()` Initialize the multi property extractor.

`<logical> isDefault()` Return true if this is the default object of this class.

`<type(varying_string)>(:) names(<integer> elementType→,<double> time→)` Return the names of the properties extracted.

`type(varying_string) objectType()` Return the type of the object.

`integer quantity()` Return the class of the extracted property.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`integer type()` Return the type of the extracted property.

`<double(:)> unitsInSI(<integer> elementType→,<double> time→)` Return the units of the properties extracted in the SI system.

nodePropertyExtractorNodeIndices

`void addInstances(\textcolor{red}{\textless type((treeNode ))\textgreater} node\arginout,\textcolor{red type((multiCounter))\textgreater} instance\arginout)` Add multiple instances of this property to a `multiCounter` object.

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((nodePropertyExtractorClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`<type(varying_string)>(:) descriptions(<double> time→)` Return descriptions of the properties extracted.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`<integer> elementCount(<double> time→)` Return the number of properties in the tuple.

`<integer(:)> extract(<type(treeNode)> node→, <double> time→, <type(multiCounter)> [instance]→)` Extract the properties from the given `node`.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`<type(varying_string)>(:) names(<double> time→)` Return the names of the properties extracted.

`type(varying_string) objectType()` Return the type of the object.

`integer quantity()` Return the class of the extracted property.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

integer type() Return the type of the extracted property.

<double(:)> unitsInSI(<double> time→) Return the units of the properties extracted in the SI system.

nodePropertyExtractorNull

void addInstances(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red type((multiCounter))\textgreater} instance\arginout) Add multiple instances of this property to a multiCounter object.

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((nodePropertyExtractorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

integer quantity() Return the class of the extracted property.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

integer type() Return the type of the extracted property.

nodePropertyExtractorProjectedDensity

`void addInstances(\textcolor{red}{\textless type((treeNode ))\textgreater} node\arginout,\textcolor{red}{\textless type((multiCounter))\textgreater} instance\arginout)` Add multiple instances of this property to a `multiCounter` object.

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\arginout,\textcolor{red}{\textless character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((nodePropertyExtractorClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`<type(varying_string)>(:) descriptions(<double> time→)` Return descriptions of the properties extracted.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\textcolor{red}{\textless logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`<integer> elementCount(<double> time→)` Return the number of properties in the tuple.

`<double(:)> extract(<type(treeNode)> node→, <double> time→, <type(multiCounter)> [instance]→)` Extract the properties from the given `node`.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDigest\argin)` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`<type(varying_string)>(:) names(<double> time→)` Return the names of the properties extracted.

`type(varying_string) objectType()` Return the type of the object.

`integer quantity()` Return the class of the extracted property.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`integer type()` Return the type of the extracted property.

`<double(:)> unitsInSI(<double> time→)` Return the units of the properties extracted in the SI system.

nodePropertyExtractorRadiiHalfLightProperties

void addInstances(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red} type((multiCounter))\textgreater} instance\arginout) Add multiple instances of this property to a multiCounter object.

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((nodePropertyExtractorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

<type(varying_string)>(:) descriptions(<double> time→) Return descriptions of the properties extracted.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

<integer> elementCount(<double> time→) Return the number of properties in the tuple.

<double(:)> extract(<type(treeNode)> node→, <double> time→, <type(multiCounter)> [instance]→) Extract the properties from the given node.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

<type(varying_string)>(:) names(<double> time→) Return the names of the properties extracted.

type(varying_string) objectType() Return the type of the object.

integer quantity() Return the class of the extracted property.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

integer type() Return the type of the extracted property.

<double(:)> unitsInSI(<double> time→) Return the units of the properties extracted in the SI system.

nodePropertyExtractorRadiusCooling

`void addInstances(\textcolor{red}{\textless type((treeNode ))\textgreater} node\arginout,\textcolor{red}{\textless type((multiCounter))\textgreater} instance\arginout)` Add multiple instances of this property to a `multiCounter` object.

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\arginout,\textcolor{red}{\textless character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((nodePropertyExtractorClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`<type(varying_string)> description()` Return a description of the property extracted.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\textcolor{red}{\textless logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`<double> extract(<type(treeNode)> node→)` Extract the property from the given `node`.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDigest\argin)` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`<type(varying_string)> name()` Return the name of the property extracted.

`type(varying_string) objectType()` Return the type of the object.

`integer quantity()` Return the class of the extracted property.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`integer type()` Return the type of the extracted property.

`<double> unitsInSI()` Return the units of the property extracted in the SI system.

nodePropertyExtractorRadiusHalfMass

void addInstances(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless type((multiCounter))\textgreater} instance\arginout) Add multiple instances of this property to a multiCounter object.

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\arginout,\textcolor{red}{\textless character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((nodePropertyExtractorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

<type(varying_string)> description() Return a description of the property extracted.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\textcolor{red}{\textless logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

<double> extract(<type(treeNode)> node→) Extract the property from the given node.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDigest\argin) Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

<type(varying_string)> name() Return the name of the property extracted.

type(varying_string) objectType() Return the type of the object.

integer quantity() Return the class of the extracted property.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

integer type() Return the type of the extracted property.

<double> unitsInSI() Return the units of the property extracted in the SI system.

nodePropertyExtractorRateCooling

`void addInstances(\textcolor{red}{\textless type((treeNode ))\textgreater} node\arginout,\textcolor{red}{\textless type((multiCounter))\textgreater} instance\arginout)` Add multiple instances of this property to a `multiCounter` object.

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\arginout,\textcolor{red}{\textless character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((nodePropertyExtractorClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`<type(varying_string)> description()` Return a description of the property extracted.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\textcolor{red}{\textless logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`<double> extract(<type(treeNode)> node→)` Extract the property from the given `node`.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDigest\argin)` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`<type(varying_string)> name()` Return the name of the property extracted.

`type(varying_string) objectType()` Return the type of the object.

`integer quantity()` Return the class of the extracted property.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`integer type()` Return the type of the extracted property.

`<double> unitsInSI()` Return the units of the property extracted in the SI system.

nodePropertyExtractorRateInfallColdMode

void addInstances(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless type((multiCounter))\textgreater} instance\arginout) Add multiple instances of this property to a multiCounter object.

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\arginout,\textcolor{red}{\textless character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((nodePropertyExtractorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

<type(varying_string)> description() Return a description of the property extracted.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\textcolor{red}{\textless logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

<double> extract(<type(treeNode)> node→) Extract the property from the given node.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDigest\argin) Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

<type(varying_string)> name() Return the name of the property extracted.

type(varying_string) objectType() Return the type of the object.

integer quantity() Return the class of the extracted property.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

integer type() Return the type of the extracted property.

<double> unitsInSI() Return the units of the property extracted in the SI system.

nodePropertyExtractorRatio

`void addInstances(\textcolor{red}{\textless type((treeNode ))\textgreater} node\arginout,\textcolor{red}{\textless type((multiCounter))\textgreater} instance\arginout)` Add multiple instances of this property to a `multiCounter` object.

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\arginout,\textcolor{red}{\textless character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((nodePropertyExtractorClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`<type(varying_string)> description()` Return a description of the property extracted.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\textcolor{red}{\textless logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`<double> extract(<type(treeNode)> node→)` Extract the property from the given `node`.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDigest\argin)` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`<type(varying_string)> name()` Return the name of the property extracted.

`type(varying_string) objectType()` Return the type of the object.

`integer quantity()` Return the class of the extracted property.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`integer type()` Return the type of the extracted property.

`<double> unitsInSI()` Return the units of the property extracted in the SI system.

nodePropertyExtractorRedshiftLastIsolated

void addInstances(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red} type((multiCounter))\textgreater} instance\arginout) Add multiple instances of this property to a multiCounter object.

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((nodePropertyExtractorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

<type(varying_string)> description() Return a description of the property extracted.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

<double> extract(<type(treeNode)> node→) Extract the property from the given node.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

<type(varying_string)> name() Return the name of the property extracted.

type(varying_string) objectType() Return the type of the object.

integer quantity() Return the class of the extracted property.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

integer type() Return the type of the extracted property.

<double> unitsInSI() Return the units of the property extracted in the SI system.

nodePropertyExtractorRotationCurve

void addInstances(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red} type((multiCounter))\textgreater} instance\arginout) Add multiple instances of this property to a multiCounter object.

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((nodePropertyExtractorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

<type(varying_string)>(:) descriptions(<double> time→) Return descriptions of the properties extracted.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

<integer> elementCount(<double> time→) Return the number of properties in the tuple.

<double(:)> extract(<type(treeNode)> node→, <double> time→, <type(multiCounter)> [instance]→) Extract the properties from the given node.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

<type(varying_string)>(:) names(<double> time→) Return the names of the properties extracted.

type(varying_string) objectType() Return the type of the object.

integer quantity() Return the class of the extracted property.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

integer type() Return the type of the extracted property.

<double(:)> unitsInSI(<double> time→) Return the units of the properties extracted in the SI system.

nodePropertyExtractorSatelliteOrbitalExtrema

void addInstances(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red} type((multiCounter))\textgreater} instance\arginout) Add multiple instances of this property to a multiCounter object.

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((nodePropertyExtractorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

<type(varying_string)>(:) descriptions(<double> time→) Return descriptions of the properties extracted.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

<integer> elementCount(<double> time→) Return the number of properties in the tuple.

<double(:)> extract(<type(treeNode)> node→, <double> time→, <type(multiCounter)> [instance]→) Extract the properties from the given node.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

<type(varying_string)>(:) names(<double> time→) Return the names of the properties extracted.

type(varying_string) objectType() Return the type of the object.

integer quantity() Return the class of the extracted property.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

integer type() Return the type of the extracted property.

<double(:)> unitsInSI(<double> time→) Return the units of the properties extracted in the SI system.

nodePropertyExtractorSatelliteStatus

`void addInstances(\textcolor{red}{\textless type((treeNode ))\textgreater} node\arginout,\textcolor{red}{\textless type((multiCounter))\textgreater} instance\arginout)` Add multiple instances of this property to a `multiCounter` object.

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\arginout,\textcolor{red}{\textless character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((nodePropertyExtractorClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`<type(varying_string)> description()` Return a description of the property extracted.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\textcolor{red}{\textless logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`<integer> extract(<type(treeNode)> node→)` Extract the property from the given `node`.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDigest\argin)` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`<type(varying_string)> name()` Return the name of the property extracted.

`type(varying_string) objectType()` Return the type of the object.

`integer quantity()` Return the class of the extracted property.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`integer type()` Return the type of the extracted property.

`<double> unitsInSI()` Return the units of the property extracted in the SI system.

nodePropertyExtractorScalar

void addInstances(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red} type((multiCounter))\textgreater} instance\arginout) Add multiple instances of this property to a multiCounter object.

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((nodePropertyExtractorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

<type(varying_string)> description() Return a description of the property extracted.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

<double> extract(<type(treeNode)> node→) Extract the property from the given node.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

<type(varying_string)> name() Return the name of the property extracted.

type(varying_string) objectType() Return the type of the object.

integer quantity() Return the class of the extracted property.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

integer type() Return the type of the extracted property.

<double> unitsInSI() Return the units of the property extracted in the SI system.

nodePropertyExtractorSpin

`void addInstances(\textcolor{red}{\textless type((treeNode ))\textgreater} node\arginout,\textcolor{red}{\textless type((multiCounter))\textgreater} instance\arginout)` Add multiple instances of this property to a `multiCounter` object.

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\arginout,\textcolor{red}{\textless character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((nodePropertyExtractorClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`<type(varying_string)> description()` Return a description of the property extracted.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\textcolor{red}{\textless logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`<double> extract(<type(treeNode)> node→)` Extract the property from the given `node`.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDigest\argin)` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`<type(varying_string)> name()` Return the name of the property extracted.

`type(varying_string) objectType()` Return the type of the object.

`integer quantity()` Return the class of the extracted property.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`integer type()` Return the type of the extracted property.

`<double> unitsInSI()` Return the units of the property extracted in the SI system.

nodePropertyExtractorSpinBullock

void addInstances(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red} type((multiCounter))\textgreater} instance\arginout) Add multiple instances of this property to a multiCounter object.

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((nodePropertyExtractorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

<type(varying_string)>(:) descriptions(<double> time→) Return descriptions of the properties extracted.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

<integer> elementCount(<double> time→) Return the number of properties in the tuple.

<double(:)> extract(<type(treeNode)> node→, <double> time→, <type(multiCounter)> [instance]→) Extract the properties from the given node.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

<type(varying_string)>(:) names(<double> time→) Return the names of the properties extracted.

type(varying_string) objectType() Return the type of the object.

integer quantity() Return the class of the extracted property.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

integer type() Return the type of the extracted property.

<double(:)> unitsInSI(<double> time→) Return the units of the properties extracted in the SI system.

nodePropertyExtractorTreeWeight

`void addInstances(\textcolor{red}{\textless type((treeNode ))\textgreater} node\arginout,\textcolor{red}{\textless type((multiCounter))\textgreater} instance\arginout)` Add multiple instances of this property to a `multiCounter` object.

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\arginout,\textcolor{red}{\textless character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((nodePropertyExtractorClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`<type(varying_string)> description()` Return a description of the property extracted.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\textcolor{red}{\textless logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`<double> extract(<type(treeNode)> node→)` Extract the property from the given `node`.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDigest\argin)` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`<type(varying_string)> name()` Return the name of the property extracted.

`type(varying_string) objectType()` Return the type of the object.

`integer quantity()` Return the class of the extracted property.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`integer type()` Return the type of the extracted property.

`<double> unitsInSI()` Return the units of the property extracted in the SI system.

nodePropertyExtractorTuple

`void addInstances(\textcolor{red}{\textless type((treeNode ))\textgreater} node\arginout,\textcolor{red}{\textless type((multiCounter))\textgreater} instance\arginout)` Add multiple instances of this property to a `multiCounter` object.

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\arginout,\textcolor{red}{\textless character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((nodePropertyExtractorClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`<type(varying_string)>(:) descriptions(<double> time→)` Return descriptions of the properties extracted.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\textcolor{red}{\textless logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`<integer> elementCount(<double> time→)` Return the number of properties in the tuple.

`<double(:)> extract(<type(treeNode)> node→, <double> time→, <type(multiCounter)> [instance]→)` Extract the properties from the given `node`.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDigest\argin)` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`<type(varying_string)>(:) names(<double> time→)` Return the names of the properties extracted.

`type(varying_string) objectType()` Return the type of the object.

`integer quantity()` Return the class of the extracted property.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`integer type()` Return the type of the extracted property.

`<double(:)> unitsInSI(<double> time→)` Return the units of the properties extracted in the SI system.

nodePropertyExtractorVelocityDispersion

`void addInstances(\textcolor{red}{\textless type((treeNode ))\textgreater} node\arginout,\textcolor{red}{\textless type((multiCounter))\textgreater} instance\arginout)` Add multiple instances of this property to a `multiCounter` object.

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\arginout,\textcolor{red}{\textless character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((nodePropertyExtractorClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`<type(varying_string)>(:) descriptions(<double> time→)` Return descriptions of the properties extracted.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\textcolor{red}{\textless logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`<integer> elementCount(<double> time→)` Return the number of properties in the tuple.

`<double(:)> extract(<type(treeNode)> node→, <double> time→, <type(multiCounter)> [instance]→)` Extract the properties from the given `node`.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDigest\argin)` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`<type(varying_string)>(:) names(<double> time→)` Return the names of the properties extracted.

`type(varying_string) objectType()` Return the type of the object.

`integer quantity()` Return the class of the extracted property.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`integer type()` Return the type of the extracted property.

`<double(:)> unitsInSI(<double> time→)` Return the units of the properties extracted in the SI system.

nodePropertyExtractorVelocityMaximum

void addInstances(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless type((multiCounter))\textgreater} instance\arginout) Add multiple instances of this property to a multiCounter object.

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\arginout,\textcolor{red}{\textless character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((nodePropertyExtractorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

<type(varying_string)> description() Return a description of the property extracted.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\textcolor{red}{\textless logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

<double> extract(<type(treeNode)> node→) Extract the property from the given node.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDigest\argin) Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

<type(varying_string)> name() Return the name of the property extracted.

type(varying_string) objectType() Return the type of the object.

integer quantity() Return the class of the extracted property.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

integer type() Return the type of the extracted property.

<double> unitsInSI() Return the units of the property extracted in the SI system.

nodePropertyExtractorVirialProperties

`void addInstances(\textcolor{red}{\textless type((treeNode ))\textgreater} node\arginout,\textcolor{red}{\textless type((multiCounter))\textgreater} instance\arginout)` Add multiple instances of this property to a `multiCounter` object.

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\arginout,\textcolor{red}{\textless character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((nodePropertyExtractorClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`<type(varying_string)>(:) descriptions(<double> time→)` Return descriptions of the properties extracted.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\textcolor{red}{\textless logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`<integer> elementCount(<double> time→)` Return the number of properties in the tuple.

`<double(:)> extract(<type(treeNode)> node→, <double> time→, <type(multiCounter)> [instance]→)` Extract the properties from the given `node`.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDigest\argin)` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`<type(varying_string)>(:) names(<double> time→)` Return the names of the properties extracted.

`type(varying_string) objectType()` Return the type of the object.

`integer quantity()` Return the class of the extracted property.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`integer type()` Return the type of the extracted property.

`<double(:)> unitsInSI(<double> time→)` Return the units of the properties extracted in the SI system.

ompIncrementalLock

<void> initialize() (Re)initialize an OpenMP incremental lock object.

<void> set() Obtain a lock on the object.

<void> unset() Release a lock on the object.

ompLock

<void> initialize() (Re)initialize an OpenMP lock object.

<logical> ownedByThread() Return true if the current thread already owns this lock.

<void> set() Obtain a lock on the object.

<void> unset() Release a lock on the object.

ompReadWriteLock

<void> initialize() (Re)initialize an OpenMP read/write lock object

<void> setRead() Obtain a read (non-blocking) lock on the object.

<void> setWrite(<logical> [haveReadLock]→) Obtain a write (blocking) lock on the object. The lock will block until all other read/write locks on the object are released and while held will prevent any read locks from being obtained. If the thread requesting the write lock already has a read lock it should set haveReadLock=.true. when calling this function.

<void> unsetRead() Release a read (non-blocking) lock on the object.

<void> unsetWrite(<logical> [haveReadLock]→) Release a write (blocking) lock on the object. If the thread releasing the write lock already had a read lock it should set haveReadLock=.true. when calling this function to ensure that read locked is retained.

operatorUnaryClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((operatorUnaryClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision operate(\textcolor{red}{\textless double precision\textgreater} x\argin) Operate on the given value.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision unoperate(\textcolor{red}{\textless double precision\textgreater} f\argin) Reverse the operation.

operatorUnaryIdentity

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((operatorUnaryClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision operate(\textcolor{red}{\textless double precision\textgreater} x\argin) Operate on the given value.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision unoperate(\textcolor{red}{\textless double precision\textgreater} f\argin) Reverse the operation.

operatorUnaryInverse

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((operatorUnaryClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision operate(\textcolor{red}{\textless double precision\textgreater} x\argin) Operate on the given value.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision unoperate(\textcolor{red}{\textless double precision\textgreater} f\argin) Reverse the operation.

operatorUnaryLogarithm

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((operatorUnaryClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision operate(\textcolor{red}{\textless double precision\textgreater} x\argin) Operate on the given value.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision unoperate(\textcolor{red}{\textless double precision\textgreater} f\argin) Reverse the operation.

outputAnalysisBlackHoleBulgeRelation

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void analyze(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\tex
integer((c_size_t))\textgreater} iOutput\argin) Analyze the given node.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

void finalize() Finalize the analysis.

<void> finalizeAnalysis() Finalize analysis of the mean function operator.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision logLikelihood() Return the log-likelihood of the analysis.

type(varying_string) objectType() Return the type of the object.

void reduce(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} reduced\arginout) Reduce the object onto another object of the class.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

<void> results(<double(:)> [binCenter]↔, <double(:,:)> [functionValue]↔, <double(:,:)> [functionCovariance]↔) Return the results of the mean function operator.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void analyze(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\tex integer((c_size_t))\textgreater} iOutput\argin) Analyze the given node.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

`void finalize()` Finalize the analysis.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision logLikelihood()` Return the log-likelihood of the analysis.

`type(varying_string) objectType()` Return the type of the object.

`void reduce(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} reduced\arginout)` Reduce the object onto another object of the class.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

outputAnalysisColorDistributionSDSS

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void analyze(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\tex integer((c_size_t))\textgreater} iOutput\argin)` Analyze the given `node`.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`void finalize()` Finalize the analysis.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision logLikelihood()` Return the log-likelihood of the analysis.

`type(varying_string) objectType()` Return the type of the object.

void reduce(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} reduced\arginout) Reduce the object onto another object of the class.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

<void> results(<double(:)> [binCenter]↔, <double(:,:)> [functionValue]↔, <double(:,:)> [functionCovariance]↔) Return the results of the volume function operator.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisConcentrationDistributionCDMCOCO

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void analyze(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\tex integer((c_size_t))\textgreater} iOutput\argin) Analyze the given node.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

void finalize() Finalize the analysis.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision logLikelihood() Return the log-likelihood of the analysis.

type(varying_string) objectType() Return the type of the object.

void reduce(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} reduced\arginout) Reduce the object onto another object of the class.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

<void> results(<double(:)> [binCenter]↔, <double(:,:)> [functionValue]↔, <double(:,:)> [functionCovariance]↔) Return the results of the volume function operator.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisConcentrationVsHaloMassCDMLudlow2016

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void analyze(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\tex integer((c_size_t))\textgreater} iOutput\argin) Analyze the given node.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

void finalize() Finalize the analysis.

<void> finalizeAnalysis() Finalize analysis of the mean function operator.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision logLikelihood() Return the log-likelihood of the analysis.

type(varying_string) objectType() Return the type of the object.

void reduce(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} reduced\arginout) Reduce the object onto another object of the class.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

<void> results(<double(:)> [binCenter]↔, <double(:,:)> [functionValue]↔, <double(:,:)> [functionCovariance]↔) Return the results of the mean function operator.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisCorrelationFunction

<void> accumulateHalo(<integer> indexOutput→, <type(treeNode)> node↔) Accumulate a halo to the correlation function.

<void> accumulateNode(<double> mass→, <integer> massType→, <integer> indexOutput→, <type(treeNode)> node↔) Accumulate a node to the correlation function.

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void analyze(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\tex integer((c_size_t))\textgreater} iOutput\argin) Analyze the given node.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

void finalize() Finalize the analysis.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision logLikelihood() Return the log-likelihood of the analysis.

type(varying_string) objectType() Return the type of the object.

void reduce(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} reduced\arginout) Reduce the object onto another object of the class.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisCorrelationFunctionHearin2013SDSS

<void> accumulateHalo(<integer> indexOutput→, <type(treeNode)> node↔) Accumulate a halo to the correlation function.

<void> accumulateNode(<double> mass→, <integer> massType→, <integer> indexOutput→, <type(treeNode)> node↔) Accumulate a node to the correlation function.

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void analyze(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\tex integer((c_size_t))\textgreater} iOutput\argin) Analyze the given node.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

void finalize() Finalize the analysis.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision logLikelihood() Return the log-likelihood of the analysis.

type(varying_string) objectType() Return the type of the object.

void reduce(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} reduced\arginout) Reduce the object onto another object of the class.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisDistributionNormalizerBinWidth

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisDistributionNormalizerClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

void normalize(\textcolor{red}{\textless double precision\textgreater} distribution\arginout,\textcolor double precision\textgreater} covariance\arginout,\textcolor{red}{\textless double precision\textgreater} propertyValueMinimum\argin,\textcolor{red}{\textless double precision\textgreater} propertyValueMaximum\argin) Normalize a distribution.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisDistributionNormalizerClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisDistributionNormalizerClass))\textgreater} destination\arginout) Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`void normalize(\textcolor{red}{\textless double precision\textgreater} distribution\arginout,\textcolor double precision\textgreater} covariance\arginout,\textcolor{red}{\textless double precision\textgreater} propertyValueMinimum\argin,\textcolor{red}{\textless double precision\textgreater} propertyValueMaximum\argin)` Normalize a distribution.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

outputAnalysisDistributionNormalizerIdentity

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((outputAnalysisDistributionNormalizerClass))\textgreater destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`void normalize(\textcolor{red}{\textless double precision\textgreater} distribution\arginout,\textcolor double precision\textgreater} covariance\arginout,\textcolor{red}{\textless double precision\textgreater} propertyValueMinimum\argin,\textcolor{red}{\textless double precision\textgreater} propertyValueMaximum\argin)` Normalize a distribution.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisDistributionNormalizerLog10ToLog

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisDistributionNormalizerClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

void normalize(\textcolor{red}{\textless double precision\textgreater} distribution\arginout,\textcolor double precision\textgreater} covariance\arginout,\textcolor{red}{\textless double precision\textgreater} propertyValueMinimum\argin,\textcolor{red}{\textless double precision\textgreater} propertyValueMaximum\argin) Normalize a distribution.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisDistributionNormalizerSequence

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((outputAnalysisDistributionNormalizerClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`void normalize(\textcolor{red}{\textless double precision\textgreater} distribution\arginout,\textcolor double precision\textgreater} covariance\arginout,\textcolor{red}{\textless double precision\textgreater} propertyValueMinimum\argin,\textcolor{red}{\textless double precision\textgreater} propertyValueMaximum\argin)` Normalize a distribution.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

outputAnalysisDistributionNormalizerUnitarity

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((outputAnalysisDistributionNormalizerClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

void normalize(\textcolor{red}{\textless double precision\textgreater} distribution\arginout,\textcolor
double precision\textgreater} covariance\arginout,\textcolor{red}{\textless double
precision\textgreater} propertyValueMinimum\argin,\textcolor{red}{\textless double
precision\textgreater} propertyValueMaximum\argin) Normalize a distribution.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisDistributionOperatorClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisDistributionOperatorClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision, dimension(size(propertyValueMinimum)) operateDistribution(\textcolor{red}{\textless
double precision\textgreater} distribution\argin,\textcolor{red}{\textless integer\textgreater}
propertyType\argin,\textcolor{red}{\textless double precision\textgreater} propertyValueMinimum\arg
double precision\textgreater} propertyValueMaximum\argin,\textcolor{red}{\textless
integer((c_size_t))\textgreater} outputIndex\argin,\textcolor{red}{\textless type((treeNode))\textg
node\arginout) Operate on a distribution to produce a distribution.

double precision, dimension(size(propertyValueMinimum)) operateScalar(\textcolor{red}{\textless double precision\textgreater} propertyValue\argin,\textcolor{red}{\textless integer\textgreater} propertyType\argin,\textcolor{red}{\textless double precision\textgreater} propertyValueMinimum\argin,\textcolor{red}{\textless double precision\textgreater} propertyValueMaximum\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} outputIndex\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Operate on a scalar to produce a distribution.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisDistributionOperatorDiskSizeInclntn

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisDistributionOperatorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\textcolor{red}{\textless logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDigest\argin) Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision, dimension(size(propertyValueMinimum)) operateDistribution(\textcolor{red}{\textless double precision\textgreater} distribution\argin,\textcolor{red}{\textless integer\textgreater} propertyType\argin,\textcolor{red}{\textless double precision\textgreater} propertyValueMinimum\argin,\textcolor{red}{\textless double precision\textgreater} propertyValueMaximum\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} outputIndex\argin,\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Operate on a distribution to produce a distribution.

double precision, dimension(size(propertyValueMinimum)) operateScalar(\textcolor{red}{\textless double precision\textgreater} propertyValue\argin,\textcolor{red}{\textless integer\textgreater} propertyType\argin,\textcolor{red}{\textless double precision\textgreater} propertyValueMinimum\arg

double precision\textgreater} propertyValueMaximum\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} outputIndex\argin,\textcolor{red}{\textless type((treeNode))\textg node\arginout) Operate on a scalar to produce a distribution.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisDistributionOperatorGrvtnlLnsng

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisDistributionOperatorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision, dimension(size(propertyValueMinimum)) operateDistribution(\textcolor{red}{\textless double precision\textgreater} distribution\argin,\textcolor{red}{\textless integer\textgreater} propertyType\argin,\textcolor{red}{\textless double precision\textgreater} propertyValueMinimum\arg double precision\textgreater} propertyValueMaximum\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} outputIndex\argin,\textcolor{red}{\textless type((treeNode))\textg node\arginout) Operate on a distribution to produce a distribution.

double precision, dimension(size(propertyValueMinimum)) operateScalar(\textcolor{red}{\textless double precision\textgreater} propertyValue\argin,\textcolor{red}{\textless integer\textgreater} propertyType\argin,\textcolor{red}{\textless double precision\textgreater} propertyValueMinimum\arg double precision\textgreater} propertyValueMaximum\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} outputIndex\argin,\textcolor{red}{\textless type((treeNode))\textg node\arginout) Operate on a scalar to produce a distribution.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisDistributionOperatorIdentity

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisDistributionOperatorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision, dimension(size(propertyValueMinimum)) operateDistribution(\textcolor{red}{\textless double precision\textgreater} distribution\argin,\textcolor{red}{\textless integer\textgreater} propertyType\argin,\textcolor{red}{\textless double precision\textgreater} propertyValueMinimum\arg double precision\textgreater} propertyValueMaximum\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} outputIndex\argin,\textcolor{red}{\textless type((treeNode))\textg node\arginout) Operate on a distribution to produce a distribution.

double precision, dimension(size(propertyValueMinimum)) operateScalar(\textcolor{red}{\textless double precision\textgreater} propertyValue\argin,\textcolor{red}{\textless integer\textgreater} propertyType\argin,\textcolor{red}{\textless double precision\textgreater} propertyValueMinimum\arg double precision\textgreater} propertyValueMaximum\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} outputIndex\argin,\textcolor{red}{\textless type((treeNode))\textg node\arginout) Operate on a scalar to produce a distribution.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisDistributionOperatorRandomError

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisDistributionOperatorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision, dimension(size(propertyValueMinimum)) operateDistribution(\textcolor{red}{\textless double precision\textgreater} distribution\argin,\textcolor{red}{\textless integer\textgreater} propertyType\argin,\textcolor{red}{\textless double precision\textgreater} propertyValueMinimum\arg double precision\textgreater} propertyValueMaximum\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} outputIndex\argin,\textcolor{red}{\textless type((treeNode))\textg node\arginout) Operate on a distribution to produce a distribution.

double precision, dimension(size(propertyValueMinimum)) operateScalar(\textcolor{red}{\textless double precision\textgreater} propertyValue\argin,\textcolor{red}{\textless integer\textgreater} propertyType\argin,\textcolor{red}{\textless double precision\textgreater} propertyValueMinimum\arg double precision\textgreater} propertyValueMaximum\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} outputIndex\argin,\textcolor{red}{\textless type((treeNode))\textg node\arginout) Operate on a scalar to produce a distribution.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

<double> rootVariance(<double> propertyValue→,<type(treeNode)> node↔) Return the root-variance to apply to the distribution.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisDistributionOperatorRandomErrorALFLF

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisDistributionOperatorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision, dimension(size(propertyValueMinimum)) operateDistribution(\textcolor{red}{\textless double precision\textgreater} distribution\argin,\textcolor{red}{\textless integer\textgreater} propertyType\argin,\textcolor{red}{\textless double precision\textgreater} propertyValueMinimum\arg double precision\textgreater} propertyValueMaximum\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} outputIndex\argin,\textcolor{red}{\textless type((treeNode))\textg node\arginout) Operate on a distribution to produce a distribution.

double precision, dimension(size(propertyValueMinimum)) operateScalar(\textcolor{red}{\textless double precision\textgreater} propertyValue\argin,\textcolor{red}{\textless integer\textgreater} propertyType\argin,\textcolor{red}{\textless double precision\textgreater} propertyValueMinimum\arg double precision\textgreater} propertyValueMaximum\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} outputIndex\argin,\textcolor{red}{\textless type((treeNode))\textg node\arginout) Operate on a scalar to produce a distribution.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

<double> rootVariance(<double> propertyValue→,<type(treeNode)> node↔) Return the root-variance to apply to the distribution.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisDistributionOperatorRandomErrorFixed

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisDistributionOperatorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision, dimension(size(propertyValueMinimum)) operateDistribution(\textcolor{red}{\textless double precision\textgreater} distribution\argin,\textcolor{red}{\textless integer\textgreater} propertyType\argin,\textcolor{red}{\textless double precision\textgreater} propertyValueMinimum\arg double precision\textgreater} propertyValueMaximum\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} outputIndex\argin,\textcolor{red}{\textless type((treeNode))\textg node\arginout) Operate on a distribution to produce a distribution.

double precision, dimension(size(propertyValueMinimum)) operateScalar(\textcolor{red}{\textless double precision\textgreater} propertyValue\argin,\textcolor{red}{\textless integer\textgreater} propertyType\argin,\textcolor{red}{\textless double precision\textgreater} propertyValueMinimum\arg double precision\textgreater} propertyValueMaximum\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} outputIndex\argin,\textcolor{red}{\textless type((treeNode))\textg node\arginout) Operate on a scalar to produce a distribution.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

<double> rootVariance(<double> propertyValue→,<type(treeNode)> node↔) Return the root-variance to apply to the distribution.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

outputAnalysisDistributionOperatorRandomErrorPlynml

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((outputAnalysisDistributionOperatorClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`double precision, dimension(size(propertyValueMinimum)) operateDistribution(\textcolor{red}{\textless double precision\textgreater} distribution\argin,\textcolor{red}{\textless integer\textgreater} propertyType\argin,\textcolor{red}{\textless double precision\textgreater} propertyValueMinimum\arg double precision\textgreater} propertyValueMaximum\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} outputIndex\argin,\textcolor{red}{\textless type((treeNode))\textg node\arginout)` Operate on a distribution to produce a distribution.

`double precision, dimension(size(propertyValueMinimum)) operateScalar(\textcolor{red}{\textless double precision\textgreater} propertyValue\argin,\textcolor{red}{\textless integer\textgreater} propertyType\argin,\textcolor{red}{\textless double precision\textgreater} propertyValueMinimum\arg double precision\textgreater} propertyValueMaximum\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} outputIndex\argin,\textcolor{red}{\textless type((treeNode))\textg node\arginout)` Operate on a scalar to produce a distribution.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`<double> rootVariance(<double> propertyValue→,<type(treeNode)> node↔)` Return the root-variance to apply to the distribution.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisDistributionOperatorRndmErrNbdyCnc

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisDistributionOperatorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision, dimension(size(propertyValueMinimum)) operateDistribution(\textcolor{red}{\textless double precision\textgreater} distribution\argin,\textcolor{red}{\textless integer\textgreater} propertyType\argin,\textcolor{red}{\textless double precision\textgreater} propertyValueMinimum\arg double precision\textgreater} propertyValueMaximum\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} outputIndex\argin,\textcolor{red}{\textless type((treeNode))\textg node\arginout) Operate on a distribution to produce a distribution.

double precision, dimension(size(propertyValueMinimum)) operateScalar(\textcolor{red}{\textless double precision\textgreater} propertyValue\argin,\textcolor{red}{\textless integer\textgreater} propertyType\argin,\textcolor{red}{\textless double precision\textgreater} propertyValueMinimum\arg double precision\textgreater} propertyValueMaximum\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} outputIndex\argin,\textcolor{red}{\textless type((treeNode))\textg node\arginout) Operate on a scalar to produce a distribution.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

<double> rootVariance(<double> propertyValue→,<type(treeNode)> node↔) Return the root-variance to apply to the distribution.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

outputAnalysisDistributionOperatorRndmErrNbodyMass

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((outputAnalysisDistributionOperatorClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`double precision, dimension(size(propertyValueMinimum)) operateDistribution(\textcolor{red}{\textless double precision\textgreater} distribution\argin,\textcolor{red}{\textless integer\textgreater} propertyType\argin,\textcolor{red}{\textless double precision\textgreater} propertyValueMinimum\arg double precision\textgreater} propertyValueMaximum\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} outputIndex\argin,\textcolor{red}{\textless type((treeNode))\textg node\arginout)` Operate on a distribution to produce a distribution.

`double precision, dimension(size(propertyValueMinimum)) operateScalar(\textcolor{red}{\textless double precision\textgreater} propertyValue\argin,\textcolor{red}{\textless integer\textgreater} propertyType\argin,\textcolor{red}{\textless double precision\textgreater} propertyValueMinimum\arg double precision\textgreater} propertyValueMaximum\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} outputIndex\argin,\textcolor{red}{\textless type((treeNode))\textg node\arginout)` Operate on a scalar to produce a distribution.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`<double> rootVariance(<double> propertyValue→,<type(treeNode)> node↔)` Return the root-variance to apply to the distribution.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisDistributionOperatorSequence

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisDistributionOperatorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision, dimension(size(propertyValueMinimum)) operateDistribution(\textcolor{red}{\textless double precision\textgreater} distribution\argin,\textcolor{red}{\textless integer\textgreater} propertyType\argin,\textcolor{red}{\textless double precision\textgreater} propertyValueMinimum\arg double precision\textgreater} propertyValueMaximum\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} outputIndex\argin,\textcolor{red}{\textless type((treeNode))\textg node\arginout) Operate on a distribution to produce a distribution.

double precision, dimension(size(propertyValueMinimum)) operateScalar(\textcolor{red}{\textless double precision\textgreater} propertyValue\argin,\textcolor{red}{\textless integer\textgreater} propertyType\argin,\textcolor{red}{\textless double precision\textgreater} propertyValueMinimum\arg double precision\textgreater} propertyValueMaximum\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} outputIndex\argin,\textcolor{red}{\textless type((treeNode))\textg node\arginout) Operate on a scalar to produce a distribution.

<void> prepend(<class(outputAnalysisDistributionOperatorClass)> operator_→) Prepend an operator to a sequence of distribution operators.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisDistributionOperatorSpinNBodyErrors

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisDistributionOperatorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision, dimension(size(propertyValueMinimum)) operateDistribution(\textcolor{red}{\textless double precision\textgreater} distribution\argin,\textcolor{red}{\textless integer\textgreater} propertyType\argin,\textcolor{red}{\textless double precision\textgreater} propertyValueMinimum\arg double precision\textgreater} propertyValueMaximum\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} outputIndex\argin,\textcolor{red}{\textless type((treeNode))\textg node\arginout) Operate on a distribution to produce a distribution.

double precision, dimension(size(propertyValueMinimum)) operateScalar(\textcolor{red}{\textless double precision\textgreater} propertyValue\argin,\textcolor{red}{\textless integer\textgreater} propertyType\argin,\textcolor{red}{\textless double precision\textgreater} propertyValueMinimum\arg double precision\textgreater} propertyValueMaximum\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} outputIndex\argin,\textcolor{red}{\textless type((treeNode))\textg node\arginout) Operate on a scalar to produce a distribution.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisGalaxySizesSDSS

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void analyze(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\tex integer((c_size_t))\textgreater} iOutput\argin) Analyze the given node.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

void finalize() Finalize the analysis.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision logLikelihood() Return the log-likelihood of the analysis.

type(varying_string) objectType() Return the type of the object.

void reduce(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} reduced\arginout) Reduce the object onto another object of the class.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

<void> results(<double(:)> [binCenter]↔, <double(:,:)> [functionValue]↔, <double(:,:)> [functionCovariance]↔) Return the results of the volume function operator.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisHIVsHaloMassRelationPadmanabhan2017

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void analyze(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\tex integer((c_size_t))\textgreater} iOutput\argin)` Analyze the given `node`.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`void finalize()` Finalize the analysis.

`<void> finalizeAnalysis()` Finalize analysis of the mean function operator.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision logLikelihood()` Return the log-likelihood of the analysis.

`type(varying_string) objectType()` Return the type of the object.

`void reduce(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} reduced\arginout)` Reduce the object onto another object of the class.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`<void> results(<double(:)> [binCenter]↔, <double(:,:)> [functionValue]↔, <double(:,:)> [functionCovariance]↔)` Return the results of the mean function operator.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

outputAnalysisLocalGroupMassFunction

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void analyze(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\tex
integer((c_size_t))\textgreater} iOutput\argin) Analyze the given node.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} destination\arginout)
Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

void finalize() Finalize the analysis.

<void> finalizeAnalysis() Finalize analysis.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision logLikelihood() Return the log-likelihood of the analysis.

type(varying_string) objectType() Return the type of the object.

void reduce(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} reduced\arginout)
Reduce the object onto another object of the class.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisLuminosityFunction

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void analyze(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\tex
integer((c_size_t))\textgreater} iOutput\argin) Analyze the given node.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`void finalize()` Finalize the analysis.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision logLikelihood()` Return the log-likelihood of the analysis.

`type(varying_string) objectType()` Return the type of the object.

`void reduce(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} reduced\arginout)` Reduce the object onto another object of the class.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`<void> results(<double(:)> [binCenter]↔, <double(:,:)> [functionValue]↔, <double(:,:)> [functionCovariance]↔)` Return the results of the volume function operator.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

outputAnalysisLuminosityFunctionGunawardhana2013SDSS

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void analyze(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\tex integer((c_size_t))\textgreater} iOutput\argin)` Analyze the given `node`.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

void finalize() Finalize the analysis.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision logLikelihood() Return the log-likelihood of the analysis.

type(varying_string) objectType() Return the type of the object.

void reduce(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} reduced\arginout) Reduce the object onto another object of the class.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

<void> results(<double(:)> [binCenter]↔, <double(:,:)> [functionValue]↔, <double(:,:)> [functionCovariance]↔) Return the results of the volume function operator.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisLuminosityFunctionHalpha

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void analyze(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\tex integer((c_size_t))\textgreater} iOutput\argin) Analyze the given node.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

void finalize() Finalize the analysis.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision logLikelihood() Return the log-likelihood of the analysis.

type(varying_string) objectType() Return the type of the object.

void reduce(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} reduced\arginout)
Reduce the object onto another object of the class.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

<void> results(<double(:)> [binCenter]↔, <double(:,:)> [functionValue]↔, <double(:,:)> [functionCovariance]↔) Return the results of the volume function operator.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisLuminosityFunctionMonteroDorta2009SDSS

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void analyze(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\tex
integer((c_size_t))\textgreater} iOutput\argin) Analyze the given node.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} destination\arginout)
Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

void finalize() Finalize the analysis.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision logLikelihood() Return the log-likelihood of the analysis.

type(varying_string) objectType() Return the type of the object.

void reduce(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} reduced\arginout) Reduce the object onto another object of the class.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

<void> results(<double(:)> [binCenter]↔, <double(:,:)> [functionValue]↔, <double(:,:)> [functionCovariance]↔) Return the results of the volume function operator.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisLuminosityFunctionSobral2013HiZELS

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void analyze(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\tex integer((c_size_t))\textgreater} iOutput\argin) Analyze the given node.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

void finalize() Finalize the analysis.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision logLikelihood() Return the log-likelihood of the analysis.

type(varying_string) objectType() Return the type of the object.

void reduce(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} reduced\arginout) Reduce the object onto another object of the class.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

<void> results(<double(:)> [binCenter]↔, <double(:,:)> [functionValue]↔, <double(:,:)> [functionCovariance]↔) Return the results of the volume function operator.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisMassFunctionHI

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void analyze(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\tex integer((c_size_t))\textgreater} iOutput\argin) Analyze the given node.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

void finalize() Finalize the analysis.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision logLikelihood() Return the log-likelihood of the analysis.

type(varying_string) objectType() Return the type of the object.

void reduce(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} reduced\arginout) Reduce the object onto another object of the class.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

<void> results(<double(:)> [binCenter]↔, <double(:,:)> [functionValue]↔, <double(:,:)> [functionCovariance]↔) Return the results of the volume function operator.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisMassFunctionHIALFALFAMartin2010

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void analyze(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\tex integer((c_size_t))\textgreater} iOutput\argin) Analyze the given node.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

void finalize() Finalize the analysis.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision logLikelihood() Return the log-likelihood of the analysis.

type(varying_string) objectType() Return the type of the object.

void reduce(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} reduced\arginout) Reduce the object onto another object of the class.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

<void> results(<double(:)> [binCenter]↔, <double(:,:)> [functionValue]↔, <double(:,:)> [functionCovariance]↔) Return the results of the volume function operator.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisMassFunctionStellar

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void analyze(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\tex integer((c_size_t))\textgreater} iOutput\argin)` Analyze the given `node`.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`void finalize()` Finalize the analysis.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision logLikelihood()` Return the log-likelihood of the analysis.

`type(varying_string) objectType()` Return the type of the object.

`void reduce(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} reduced\arginout)` Reduce the object onto another object of the class.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`<void> results(<double(:)> [binCenter]↔, <double(:,:)> [functionValue]↔, <double(:,:)> [functionCovariance]↔)` Return the results of the volume function operator.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

outputAnalysisMassFunctionStellarBaldry2012GAMA

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void analyze(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\tex
integer((c_size_t))\textgreater} iOutput\argin) Analyze the given node.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} destination\arginout)
Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

void finalize() Finalize the analysis.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision logLikelihood() Return the log-likelihood of the analysis.

type(varying_string) objectType() Return the type of the object.

void reduce(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} reduced\arginout)
Reduce the object onto another object of the class.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

<void> results(<double(:)> [binCenter]↔, <double(:,:)> [functionValue]↔, <double(:,:)>
[functionCovariance]↔) Return the results of the volume function operator.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisMassFunctionStellarBernardi2013SDSS

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void analyze(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\tex
integer((c_size_t))\textgreater} iOutput\argin) Analyze the given node.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} destination\arginout)
Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

void finalize() Finalize the analysis.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision logLikelihood() Return the log-likelihood of the analysis.

type(varying_string) objectType() Return the type of the object.

void reduce(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} reduced\arginout)
Reduce the object onto another object of the class.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

<void> results(<double(:)> [binCenter]↔, <double(:,:)> [functionValue]↔, <double(:,:)>
[functionCovariance]↔) Return the results of the volume function operator.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisMassFunctionStellarPRIMUS

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void analyze(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\tex integer((c_size_t))\textgreater} iOutput\argin)` Analyze the given `node`.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`void finalize()` Finalize the analysis.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision logLikelihood()` Return the log-likelihood of the analysis.

`type(varying_string) objectType()` Return the type of the object.

`void reduce(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} reduced\arginout)` Reduce the object onto another object of the class.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`<void> results(<double(:)> [binCenter]↔, <double(:,:)> [functionValue]↔, <double(:,:)> [functionCovariance]↔)` Return the results of the volume function operator.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

outputAnalysisMassFunctionStellarSDSS

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void analyze(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\tex integer((c_size_t))\textgreater} iOutput\argin)` Analyze the given `node`.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`void finalize()` Finalize the analysis.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision logLikelihood()` Return the log-likelihood of the analysis.

`type(varying_string) objectType()` Return the type of the object.

`void reduce(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} reduced\arginout)` Reduce the object onto another object of the class.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`<void> results(<double(:)> [binCenter]↔, <double(:,:)> [functionValue]↔, <double(:,:)> [functionCovariance]↔)` Return the results of the volume function operator.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

outputAnalysisMassFunctionStellarUKIDSSUDS

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void analyze(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\tex
integer((c_size_t))\textgreater} iOutput\argin) Analyze the given node.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} destination\arginout)
Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

void finalize() Finalize the analysis.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision logLikelihood() Return the log-likelihood of the analysis.

type(varying_string) objectType() Return the type of the object.

void reduce(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} reduced\arginout)
Reduce the object onto another object of the class.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

<void> results(<double(:)> [binCenter]↔, <double(:,:)> [functionValue]↔, <double(:,:)>
[functionCovariance]↔) Return the results of the volume function operator.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisMassFunctionStellarULTRAVISTA

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void analyze(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\tex
integer((c_size_t))\textgreater} iOutput\argin) Analyze the given node.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} destination\arginout)
Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

void finalize() Finalize the analysis.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision logLikelihood() Return the log-likelihood of the analysis.

type(varying_string) objectType() Return the type of the object.

void reduce(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} reduced\arginout)
Reduce the object onto another object of the class.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

<void> results(<double(:)> [binCenter]↔, <double(:,:)> [functionValue]↔, <double(:,:)>
[functionCovariance]↔) Return the results of the volume function operator.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisMassFunctionStellarVIPERS

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void analyze(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\tex
integer((c_size_t))\textgreater} iOutput\argin) Analyze the given node.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} destination\arginout)
Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

void finalize() Finalize the analysis.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision logLikelihood() Return the log-likelihood of the analysis.

type(varying_string) objectType() Return the type of the object.

void reduce(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} reduced\arginout)
Reduce the object onto another object of the class.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

<void> results(<double(:)> [binCenter]↔, <double(:,:)> [functionValue]↔, <double(:,:)>
[functionCovariance]↔) Return the results of the volume function operator.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisMassFunctionStellarZFOURGE

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void analyze(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\tex integer((c_size_t))\textgreater} iOutput\argin)` Analyze the given `node`.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`void finalize()` Finalize the analysis.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision logLikelihood()` Return the log-likelihood of the analysis.

`type(varying_string) objectType()` Return the type of the object.

`void reduce(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} reduced\arginout)` Reduce the object onto another object of the class.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`<void> results(<double(:)> [binCenter]↔, <double(:,:)> [functionValue]↔, <double(:,:)> [functionCovariance]↔)` Return the results of the volume function operator.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

outputAnalysisMassMetallicityAndrews2013

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void analyze(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\tex
integer((c_size_t))\textgreater} iOutput\argin) Analyze the given node.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} destination\arginout)
Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

void finalize() Finalize the analysis.

<void> finalizeAnalysis() Finalize analysis of the mean function operator.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision logLikelihood() Return the log-likelihood of the analysis.

type(varying_string) objectType() Return the type of the object.

void reduce(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} reduced\arginout)
Reduce the object onto another object of the class.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

<void> results(<double(:)> [binCenter]↔, <double(:,:)> [functionValue]↔, <double(:,:)>
[functionCovariance]↔) Return the results of the mean function operator.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisMassMetallicityBlanc2017

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void analyze(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\tex
integer((c_size_t))\textgreater} iOutput\argin) Analyze the given node.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} destination\arginout)
Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

void finalize() Finalize the analysis.

<void> finalizeAnalysis() Finalize analysis of the mean function operator.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision logLikelihood() Return the log-likelihood of the analysis.

type(varying_string) objectType() Return the type of the object.

void reduce(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} reduced\arginout)
Reduce the object onto another object of the class.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

<void> results(<double(:)> [binCenter]↔, <double(:,:)> [functionValue]↔, <double(:,:)>
[functionCovariance]↔) Return the results of the mean function operator.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisMeanFunction1D

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void analyze(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\tex
integer((c_size_t))\textgreater} iOutput\argin) Analyze the given node.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} destination\arginout)
Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

void finalize() Finalize the analysis.

<void> finalizeAnalysis() Finalize analysis of the mean function operator.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision logLikelihood() Return the log-likelihood of the analysis.

type(varying_string) objectType() Return the type of the object.

void reduce(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} reduced\arginout)
Reduce the object onto another object of the class.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

<void> results(<double(:)> [binCenter]↔, <double(:,:)> [functionValue]↔, <double(:,:)>
[functionCovariance]↔) Return the results of the mean function operator.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisMolecularRatioClass

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((outputAnalysisMolecularRatioClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`double precision ratio(\textcolor{red}{\textless double precision\textgreater} massISM\argin,\textcolor type((treeNode))\textgreater} node\arginout)` Return the molecular ratio, $R_{\rm mol} = M_{\rm H_2}/M_{\rm HI}$.

`double precision ratioScatter(\textcolor{red}{\textless double precision\textgreater} massISM\argin,\te type((treeNode))\textgreater} node\arginout)` Return the scatter in logarithmic molecular ratio, $\log_{10} R_{\rm mol} = \log_{10}(M_{\rm H_2}/M_{\rm HI})$.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

outputAnalysisMolecularRatioObreschkow2009

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((outputAnalysisMolecularRatioClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`double precision ratio(\textcolor{red}{\textless double precision\textgreater} massISM\argin,\textcolor type((treeNode))\textgreater} node\arginout)` Return the molecular ratio, $R_{\rm mol} = M_{\rm H_2}/M_{\rm HI}$.

`double precision ratioScatter(\textcolor{red}{\textless double precision\textgreater} massISM\argin,\te type((treeNode))\textgreater} node\arginout)` Return the scatter in logarithmic molecular ratio, $\log_{10} R_{\rm mol} = \log_{10}(M_{\rm H_2}/M_{\rm HI})$.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

outputAnalysisMorphologicalFractionGAMAMoffett2016

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void analyze(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\tex integer((c_size_t))\textgreater} iOutput\argin)` Analyze the given `node`.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`void finalize()` Finalize the analysis.

`<void> finalizeAnalysis()` Finalize analysis of the mean function operator.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision logLikelihood() Return the log-likelihood of the analysis.

type(varying_string) objectType() Return the type of the object.

void reduce(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} reduced\arginout)
Reduce the object onto another object of the class.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

<void> results(<double(:)> [binCenter]↔, <double(:,:)> [functionValue]↔, <double(:,:)> [functionCovariance]↔) Return the results of the mean function operator.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisMulti

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void analyze(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\tex
integer((c_size_t))\textgreater} iOutput\argin) Analyze the given node.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} destination\arginout)
Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

void finalize() Finalize the analysis.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision logLikelihood() Return the log-likelihood of the analysis.

type(varying_string) objectType() Return the type of the object.

void reduce(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} reduced\arginout) Reduce the object onto another object of the class.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisNull

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void analyze(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\tex integer((c_size_t))\textgreater} iOutput\argin) Analyze the given node.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

void finalize() Finalize the analysis.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision logLikelihood() Return the log-likelihood of the analysis.

type(varying_string) objectType() Return the type of the object.

void reduce(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} reduced\arginout) Reduce the object onto another object of the class.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisPropertyOperatorAntiLog10

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisPropertyOperatorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision operate(\textcolor{red}{\textless double precision\textgreater} propertyValue\argin,\t type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater} propertyType\arginout,\textcolor{red}{\textless integer((c_size_t))\textgreater} outputIndex\argin) Operate on the given property.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisPropertyOperatorBoolean

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisPropertyOperatorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision operate(\textcolor{red}{\textless double precision\textgreater} propertyValue\argin,\t type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater} propertyType\arginout,\textcolor{red}{\textless integer((c_size_t))\textgreater} outputIndex\argin) Operate on the given property.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisPropertyOperatorClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisPropertyOperatorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision operate(\textcolor{red}{\textless double precision\textgreater} propertyValue\argin,\t
type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater}
propertyType\arginout,\textcolor{red}{\textless integer((c_size_t))\textgreater} outputIndex\argin)
Operate on the given property.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisPropertyOperatorCsmlgyAnglrDstnc

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisPropertyOperatorClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision operate(\textcolor{red}{\textless double precision\textgreater} propertyValue\argin,\t
type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater}
propertyType\arginout,\textcolor{red}{\textless integer((c_size_t))\textgreater} outputIndex\argin)
Operate on the given property.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisPropertyOperatorCsmlgyLmnstyDstnc

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisPropertyOperatorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision operate(\textcolor{red}{\textless double precision\textgreater} propertyValue\argin,\t type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater} propertyType\arginout,\textcolor{red}{\textless integer((c_size_t))\textgreater} outputIndex\argin) Operate on the given property.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisPropertyOperatorFilterHighPass

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((outputAnalysisPropertyOperatorClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`double precision operate(\textcolor{red}{\textless double precision\textgreater} propertyValue\argin,\t type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater} propertyType\arginout,\textcolor{red}{\textless integer((c_size_t))\textgreater} outputIndex\argin)` Operate on the given property.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

outputAnalysisPropertyOperatorHIMass

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((outputAnalysisPropertyOperatorClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision operate(\textcolor{red}{\textless double precision\textgreater} propertyValue\argin,\t
type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater}
propertyType\arginout,\textcolor{red}{\textless integer((c_size_t))\textgreater} outputIndex\argin)
Operate on the given property.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisPropertyOperatorIdentity

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisPropertyOperatorClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision operate(\textcolor{red}{\textless double precision\textgreater} propertyValue\argin,\t
type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater}
propertyType\arginout,\textcolor{red}{\textless integer((c_size_t))\textgreater} outputIndex\argin)
Operate on the given property.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisPropertyOperatorLog10

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisPropertyOperatorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision operate(\textcolor{red}{\textless double precision\textgreater} propertyValue\argin,\t type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater} propertyType\arginout,\textcolor{red}{\textless integer((c_size_t))\textgreater} outputIndex\argin) Operate on the given property.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisPropertyOperatorMagnitude

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((outputAnalysisPropertyOperatorClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`double precision operate(\textcolor{red}{\textless double precision\textgreater} propertyValue\argin,\t type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater} propertyType\arginout,\textcolor{red}{\textless integer((c_size_t))\textgreater} outputIndex\argin)` Operate on the given property.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

outputAnalysisPropertyOperatorMetallicity12LogNH

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((outputAnalysisPropertyOperatorClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision operate(\textcolor{red}{\textless double precision\textgreater} propertyValue\argin,\t
type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater}
propertyType\arginout,\textcolor{red}{\textless integer((c_size_t))\textgreater} outputIndex\argin)
Operate on the given property.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisPropertyOperatorMinMax

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisPropertyOperatorClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision operate(\textcolor{red}{\textless double precision\textgreater} propertyValue\argin,\t
type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater}
propertyType\arginout,\textcolor{red}{\textless integer((c_size_t))\textgreater} outputIndex\argin)
Operate on the given property.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisPropertyOperatorMultiply

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisPropertyOperatorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision operate(\textcolor{red}{\textless double precision\textgreater} propertyValue\argin,\t type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater} propertyType\arginout,\textcolor{red}{\textless integer((c_size_t))\textgreater} outputIndex\argin) Operate on the given property.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisPropertyOperatorNormal

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((outputAnalysisPropertyOperatorClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`double precision operate(\textcolor{red}{\textless double precision\textgreater} propertyValue\argin,\t type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater} propertyType\arginout,\textcolor{red}{\textless integer((c_size_t))\textgreater} outputIndex\argin)` Operate on the given property.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

outputAnalysisPropertyOperatorSequence

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((outputAnalysisPropertyOperatorClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision operate(\textcolor{red}{\textless double precision\textgreater} propertyValue\argin,\t
type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater}
propertyType\arginout,\textcolor{red}{\textless integer((c_size_t))\textgreater} outputIndex\argin)
Operate on the given property.

<void> prepend(<class(outputAnalysisPropertyOperatorClass)> operator_→) Prepend an operator to a sequence of property operators.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisPropertyOperatorSquare

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisPropertyOperatorClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision operate(\textcolor{red}{\textless double precision\textgreater} propertyValue\argin,\t
type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater}
propertyType\arginout,\textcolor{red}{\textless integer((c_size_t))\textgreater} outputIndex\argin)
Operate on the given property.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisPropertyOperatorSquareRoot

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisPropertyOperatorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision operate(\textcolor{red}{\textless double precision\textgreater} propertyValue\argin,\t type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater} propertyType\arginout,\textcolor{red}{\textless integer((c_size_t))\textgreater} outputIndex\argin) Operate on the given property.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisPropertyOperatorSystmtcPolynomial

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisPropertyOperatorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision operate(\textcolor{red}{\textless double precision\textgreater} propertyValue\argin,\t type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless integer\textgreater} propertyType\arginout,\textcolor{red}{\textless integer((c_size_t))\textgreater} outputIndex\argin) Operate on the given property.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisScatterFunction1D

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void analyze(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\tex integer((c_size_t))\textgreater} iOutput\argin) Analyze the given node.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} destination\arginout) Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`void finalize()` Finalize the analysis.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision logLikelihood()` Return the log-likelihood of the analysis.

`type(varying_string) objectType()` Return the type of the object.

`void reduce(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} reduced\arginout)` Reduce the object onto another object of the class.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

outputAnalysisSpinDistributionBett2007

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void analyze(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\tex integer((c_size_t))\textgreater} iOutput\argin)` Analyze the given `node`.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`void finalize()` Finalize the analysis.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision logLikelihood() Return the log-likelihood of the analysis.

type(varying_string) objectType() Return the type of the object.

void reduce(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} reduced\arginout) Reduce the object onto another object of the class.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

<void> results(<double(:)> [binCenter]↔, <double(:,:)> [functionValue]↔, <double(:,:)> [functionCovariance]↔) Return the results of the volume function operator.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisStellarVsHaloMassRelationLeauthaud2012

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void analyze(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\tex integer((c_size_t))\textgreater} iOutput\argin) Analyze the given node.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

void finalize() Finalize the analysis.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision logLikelihood() Return the log-likelihood of the analysis.

type(varying_string) objectType() Return the type of the object.

`void reduce(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} reduced\arginout)` Reduce the object onto another object of the class.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

outputAnalysisVolumeFunction1D

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void analyze(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\tex integer((c_size_t))\textgreater} iOutput\argin)` Analyze the given `node`.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`void finalize()` Finalize the analysis.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision logLikelihood()` Return the log-likelihood of the analysis.

`type(varying_string) objectType()` Return the type of the object.

`void reduce(\textcolor{red}{\textless class((outputAnalysisClass))\textgreater} reduced\arginout)` Reduce the object onto another object of the class.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

<void> results(<double(:)> [binCenter]↔, <double(:,:)> [functionValue]↔, <double(:,:)> [functionCovariance]↔) Return the results of the volume function operator.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisWeightOperatorClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisWeightOperatorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision operate(\textcolor{red}{\textless double precision\textgreater} weightValue\argin,\tex type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} propertyValue\argin,\textcolor{red}{\textless double precision\textgreater} propertyValueIntrinsic\ integer\textgreater} propertyType\argin,\textcolor{red}{\textless integer\textgreater} propertyQuantity\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} outputIndex\argin Operate on the given weight.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisWeightOperatorCsmlgyVolume

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisWeightOperatorClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision operate(\textcolor{red}{\textless double precision\textgreater} weightValue\argin,\tex
type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater}
propertyValue\argin,\textcolor{red}{\textless double precision\textgreater} propertyValueIntrinsic\
integer\textgreater} propertyType\argin,\textcolor{red}{\textless integer\textgreater}
propertyQuantity\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} outputIndex\argin
Operate on the given weight.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisWeightOperatorFilterHighPass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisWeightOperatorClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`double precision operate(\textcolor{red}{\textless double precision\textgreater} weightValue\argin,\tex type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} propertyValue\argin,\textcolor{red}{\textless double precision\textgreater} propertyValueIntrinsic\ integer\textgreater} propertyType\argin,\textcolor{red}{\textless integer\textgreater} propertyQuantity\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} outputIndex\argin` Operate on the given weight.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

outputAnalysisWeightOperatorIdentity

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((outputAnalysisWeightOperatorClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

double precision operate(\textcolor{red}{\textless double precision\textgreater} weightValue\argin,\tex type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} propertyValue\argin,\textcolor{red}{\textless double precision\textgreater} propertyValueIntrinsic\ integer\textgreater} propertyType\argin,\textcolor{red}{\textless integer\textgreater} propertyQuantity\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} outputIndex\argin Operate on the given weight.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisWeightOperatorNbodyMass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisWeightOperatorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision operate(\textcolor{red}{\textless double precision\textgreater} weightValue\argin,\tex type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} propertyValue\argin,\textcolor{red}{\textless double precision\textgreater} propertyValueIntrinsic\ integer\textgreater} propertyType\argin,\textcolor{red}{\textless integer\textgreater} propertyQuantity\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} outputIndex\argin Operate on the given weight.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

<double> rootVariance(<type(treeNode)> node↔, <double> propertyValue→, <double> propertyValueIntrinsi
<integer> propertyType→, <integer> propertyQuantity→, <integer(c_size_t)> outputIndex→)
Return the root-variance to use in the weight operator.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisWeightOperatorNormal

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisWeightOperatorClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision operate(\textcolor{red}{\textless double precision\textgreater} weightValue\argin,\tex
type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater}
propertyValue\argin,\textcolor{red}{\textless double precision\textgreater} propertyValueIntrinsic\
integer\textgreater} propertyType\argin,\textcolor{red}{\textless integer\textgreater}
propertyQuantity\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} outputIndex\argin
Operate on the given weight.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

<double> rootVariance(<type(treeNode)> node↔, <double> propertyValue→, <double> propertyValueIntrinsi
<integer> propertyType→, <integer> propertyQuantity→, <integer(c_size_t)> outputIndex→)
Return the root-variance to use in the weight operator.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisWeightOperatorProperty

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisWeightOperatorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision operate(\textcolor{red}{\textless double precision\textgreater} weightValue\argin,\tex type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater} propertyValue\argin,\textcolor{red}{\textless double precision\textgreater} propertyValueIntrinsic\ integer\textgreater} propertyType\argin,\textcolor{red}{\textless integer\textgreater} propertyQuantity\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} outputIndex\argin Operate on the given weight.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputAnalysisWeightOperatorSequence

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((outputAnalysisWeightOperatorClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision operate(\textcolor{red}{\textless double precision\textgreater} weightValue\argin,\tex
type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless double precision\textgreater}
propertyValue\argin,\textcolor{red}{\textless double precision\textgreater} propertyValueIntrinsic\
integer\textgreater} propertyType\argin,\textcolor{red}{\textless integer\textgreater}
propertyQuantity\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} outputIndex\argin
Operate on the given weight.

<void> prepend(<class(outputAnalysisWeightOperatorClass)> operator_→) Prepend an operator to a sequence of weight operators.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

outputTimesClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

integer(c_size_t) count() Return the number of output times.

`void deepCopy(\textcolor{red}{\textless class((outputTimesClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`integer(c_size_t) index(\textcolor{red}{\textless double precision\textgreater} time \argin,\textcolor{ logical\textgreater} findClosest\argin)` Return the index of the output at the given `time`. If `findClosest` is given and is true then the closest matching output is returned.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`double precision redshift(\textcolor{red}{\textless integer((c_size_t))\textgreater} indexOutput\argin)` Return the output redshift index by `indexOutput`.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision time(\textcolor{red}{\textless integer((c_size_t))\textgreater} indexOutput\argin)` Return the output time index by `indexOutput`.

`double precision timeNext(\textcolor{red}{\textless double precision\textgreater} timeCurrent\argin,\te integer((c_size_t))\textgreater} indexOutput\argout)` Given a `time`, return the time of the next output, and (optionally) the index of that output.

`double precision timePrevious(\textcolor{red}{\textless double precision\textgreater} timeCurrent\argin` Given a `time`, return the time of the previous output.

outputTimesList

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`integer(c_size_t) count()` Return the number of output times.

`void deepCopy(\textcolor{red}{\textless class((outputTimesClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`integer(c_size_t) index(\textcolor{red}{\textless double precision\textgreater} time \argin,\textcolor{ logical\textgreater} findClosest\argin)` Return the index of the output at the given `time`. If `findClosest` is given and is true then the closest matching output is returned.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`double precision redshift(\textcolor{red}{\textless integer((c_size_t))\textgreater} indexOutput\argin)` Return the output redshift index by `indexOutput`.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision time(\textcolor{red}{\textless integer((c_size_t))\textgreater} indexOutput\argin)` Return the output time index by `indexOutput`.

`double precision timeNext(\textcolor{red}{\textless double precision\textgreater} timeCurrent\argin,\te integer((c_size_t))\textgreater} indexOutput\argout)` Given a `time`, return the time of the next output, and (optionally) the index of that output.

`double precision timePrevious(\textcolor{red}{\textless double precision\textgreater} timeCurrent\argin` Given a `time`, return the time of the previous output.

polygon

`<logical> pointIncluded(<double(3)> point→)` Return true if the given point lives inside the MANGLE polygon.

polynomialIterator

<integer(:)> counter() Return an incremental counter (i.e. begins at 0 and increases by 1 on each iteration).

<integer(:)> currentOrder() Return the current order of the polynomial coefficient.

<integer(:)> index(<integer(:)> i→) Return the i^{th} index of the polynomial coefficient.

<logical> iterate() Move to the next iteration of polynomial coefficient indices. Returns true if successful. If no more iterations are available, returns false.

<void> reset() Reset the iterator object to the start of its sequence.

posteriorSampleConvergenceClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

integer convergedAtStep() Returns the step at which the sampling reached convergence.

void deepCopy(\textcolor{red}{\textless class((posteriorSampleConvergenceClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

logical isConverged(\textcolor{red}{\textless class((posteriorSampleStateClass))\textgreater} simulationState\arginout,\textcolor{red}{\textless double precision\textgreater} logLikelihood\argi Returns true if the posterior sampling is deemed to be converged.

<logical> isDefault() Return true if this is the default object of this class.

void logReport(\textcolor{red}{\textless integer\textgreater} fileUnit\argin) Log a report on convergence state to the given file unit.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void reset() Reset the convergence calculation.

logical stateIsOutlier(\textcolor{red}{\textless integer\textgreater} stateIndex\argin) Return true if the specified state is deemed to be an outlier.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

posteriorSampleConvergenceGelmanRubin

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`integer convergedAtStep()` Returns the step at which the sampling reached convergence.

`<double> convergenceMeasure()` Return the current convergence measure, $\hat{R}$.

`<double> convergenceMeasureTarget()` Return the target convergence measure, $\hat{R}$.

`void deepCopy(\textcolor{red}{\textless class((posteriorSampleConvergenceClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`logical isConverged(\textcolor{red}{\textless class((posteriorSampleStateClass))\textgreater} simulationState\arginout,\textcolor{red}{\textless double precision\textgreater} logLikelihood\argi` Returns true if the posterior sampling is deemed to be converged.

`<logical> isDefault()` Return true if this is the default object of this class.

`void logReport(\textcolor{red}{\textless integer\textgreater} fileUnit\argin)` Log a report on convergence state to the given file unit.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void reset()` Reset the convergence calculation.

`logical stateIsOutlier(\textcolor{red}{\textless integer\textgreater} stateIndex\argin)` Return true if the specified state is deemed to be an outlier.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

posteriorSampleConvergenceLikelihoodThreshold

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`integer convergedAtStep()` Returns the step at which the sampling reached convergence.

`void deepCopy(\textcolor{red}{\textless class((posteriorSampleConvergenceClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`logical isConverged(\textcolor{red}{\textless class((posteriorSampleStateClass))\textgreater} simulationState\arginout,\textcolor{red}{\textless double precision\textgreater} logLikelihood\argi` Returns true if the posterior sampling is deemed to be converged.

`<logical> isDefault()` Return true if this is the default object of this class.

`void logReport(\textcolor{red}{\textless integer\textgreater} fileUnit\argin)` Log a report on convergence state to the given file unit.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void reset()` Reset the convergence calculation.

`logical stateIsOutlier(\textcolor{red}{\textless integer\textgreater} stateIndex\argin)` Return true if the specified state is deemed to be an outlier.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

posteriorSampleConvergenceNever

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`integer convergedAtStep()` Returns the step at which the sampling reached convergence.

`void deepCopy(\textcolor{red}{\textless class((posteriorSampleConvergenceClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`logical isConverged(\textcolor{red}{\textless class((posteriorSampleStateClass))\textgreater} simulationState\arginout,\textcolor{red}{\textless double precision\textgreater} logLikelihood\argi` Returns true if the posterior sampling is deemed to be converged.

`<logical> isDefault()` Return true if this is the default object of this class.

`void logReport(\textcolor{red}{\textless integer\textgreater} fileUnit\argin)` Log a report on convergence state to the given file unit.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void reset()` Reset the convergence calculation.

`logical stateIsOutlier(\textcolor{red}{\textless integer\textgreater} stateIndex\argin)` Return true if the specified state is deemed to be an outlier.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

posteriorSampleDffrntlEvltnProposalSizeAdaptive

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((posteriorSampleDffrntlEvltnProposalSizeClass))\textgreat destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision gamma(\textcolor{red}{\textless class((posteriorSampleStateClass ))\textgreater} simulationState\arginout,\textcolor{red}{\textless class((posteriorSampleConvergenceClass))\textgre simulationConvergence\arginout)` Sample from the jump distribution.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

posteriorSampleDffrntlEvltnProposalSizeClass

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((posteriorSampleDffrntlEvltnProposalSizeClass))\textgreat destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

double precision gamma(\textcolor{red}{\textless class((posteriorSampleStateClass))\textgreater} simulationState\arginout,\textcolor{red}{\textless class((posteriorSampleConvergenceClass))\textgre simulationConvergence\arginout) Sample from the jump distribution.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

posteriorSampleDffrntlEvltnProposalSizeFixed

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((posteriorSampleDffrntlEvltnProposalSizeClass))\textgreat destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision gamma(\textcolor{red}{\textless class((posteriorSampleStateClass))\textgreater} simulationState\arginout,\textcolor{red}{\textless class((posteriorSampleConvergenceClass))\textgre simulationConvergence\arginout) Sample from the jump distribution.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

posteriorSampleDffrntlEvltnPrpslSzTmpExpAdaptive

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((posteriorSampleDffrntlEvltnPrpslSzTmpExpClass))\textgrea destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision exponent(\textcolor{red}{\textless class((posteriorSampleStateClass))\textgreater} temperedStates\arginout,\textcolor{red}{\textless double precision\textgreater} temperatures\argin, class((posteriorSampleStateClass))\textgreater} simulationState\arginout,\textcolor{red}{\textless class((posteriorSampleConvergenceClass))\textgreater} simulationConvergence\arginout) Return the temperature exponent.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

posteriorSampleDffrntlEvltnPrpslSzTmpExpClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((posteriorSampleDffrntlEvltnPrpslSzTmpExpClass))\textgrea
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision exponent(\textcolor{red}{\textless class((posteriorSampleStateClass))\textgreater}
temperedStates\arginout,\textcolor{red}{\textless double precision\textgreater} temperatures\argin,
class((posteriorSampleStateClass))\textgreater} simulationState\arginout,\textcolor{red}{\textless
class((posteriorSampleConvergenceClass))\textgreater} simulationConvergence\arginout)
Return the temperature exponent.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

posteriorSampleDffrntlEvltnPrpslSzTmpExpFixed

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((posteriorSampleDffrntlEvltnPrpslSzTmpExpClass))\textgrea
destination\arginout) Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision exponent(\textcolor{red}{\textless class((posteriorSampleStateClass))\textgreater} temperedStates\arginout,\textcolor{red}{\textless double precision\textgreater} temperatures\argin, class((posteriorSampleStateClass))\textgreater} simulationState\arginout,\textcolor{red}{\textless class((posteriorSampleConvergenceClass))\textgreater} simulationConvergence\arginout)` Return the temperature exponent.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

posteriorSampleDffrntlEvltnRandomJumpAdaptive

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((posteriorSampleDffrntlEvltnRandomJumpClass))\textgreater destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

double precision, dimension(size(modelParameters_)) sample(\textcolor{red}{\textless type((modelParamet
))\textgreater} modelParameters_\argin,\textcolor{red}{\textless class((posteriorSampleStateClass))
simulationState\arginout) Sample from the jump distribution.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

posteriorSampleDffrntlEvltnRandomJumpClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((posteriorSampleDffrntlEvltnRandomJumpClass))\textgreater
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

double precision, dimension(size(modelParameters_)) sample(\textcolor{red}{\textless type((modelParamet
))\textgreater} modelParameters_\argin,\textcolor{red}{\textless class((posteriorSampleStateClass))
simulationState\arginout) Sample from the jump distribution.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

posteriorSampleDffrntlEvltnRandomJumpSimple

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((posteriorSampleDffrntlEvltnRandomJumpClass))\textgreater destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`double precision, dimension(size(modelParameters_)) sample(\textcolor{red}{\textless type((modelParamet ))\textgreater} modelParameters_\argin,\textcolor{red}{\textless class((posteriorSampleStateClass)) simulationState\arginout)` Sample from the jump distribution.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

posteriorSampleLikelihoodClass

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((posteriorSampleLikelihoodClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

double precision evaluate(\textcolor{red}{\textless class((posteriorSampleStateClass))\textgreater} simulationState\arginout,\textcolor{red}{\textless type((modelParameterList))\textgreater} modelParametersActive_\argin,\textcolor{red}{\textless type((modelParameterList))\textgreater} modelParametersInactive_\argin,\textcolor{red}{\textless class((posteriorSampleConvergenceClass))\t simulationConvergence\arginout,\textcolor{red}{\textless double precision\textgreater} temperature\argin,\textcolor{red}{\textless double precision\textgreater} logLikelihoodCurrent\argi double precision\textgreater} logPriorCurrent\argin,\textcolor{red}{\textless double precision\textgreater} logPriorProposed\argin,\textcolor{red}{\textless real\textgreater} timeEvaluate\arginout,\textcolor{red}{\textless double precision\textgreater} logLikelihoodVariance logical\textgreater} forceAcceptance\arginout) Evaluate the likelihood.

void functionChanged() Respond to possible changes in the likelihood function.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void restore(\textcolor{red}{\textless double precision\textgreater} simulationState\argin,\textcolor{r double precision\textgreater} logLikelihood\argin) Log a report on convergence state to the given file unit.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

logical willEvaluate(\textcolor{red}{\textless class((posteriorSampleStateClass))\textgreater} simulationState\arginout,\textcolor{red}{\textless type((modelParameterList))\textgreater} modelParameters_\argin,\textcolor{red}{\textless class((posteriorSampleConvergenceClass))\textgreat simulationConvergence\arginout,\textcolor{red}{\textless double precision\textgreater} temperature\argin,\textcolor{red}{\textless double precision\textgreater} logLikelihoodCurrent\argi double precision\textgreater} logPriorCurrent\argin,\textcolor{red}{\textless double precision\textgreater} logPriorProposed\argin) Returns true if the likelihood will be evaluated for this state.

posteriorSampleLikelihoodGalaxyPopulation

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((posteriorSampleLikelihoodClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision evaluate(\textcolor{red}{\textless class((posteriorSampleStateClass))\textgreater} simulationState\arginout,\textcolor{red}{\textless type((modelParameterList))\textgreater} modelParametersActive_\argin,\textcolor{red}{\textless type((modelParameterList))\textgreater} modelParametersInactive_\argin,\textcolor{red}{\textless class((posteriorSampleConvergenceClass))\t simulationConvergence\arginout,\textcolor{red}{\textless double precision\textgreater} temperature\argin,\textcolor{red}{\textless double precision\textgreater} logLikelihoodCurrent\argi double precision\textgreater} logPriorCurrent\argin,\textcolor{red}{\textless double precision\textgreater} logPriorProposed\argin,\textcolor{red}{\textless real\textgreater} timeEvaluate\arginout,\textcolor{red}{\textless double precision\textgreater} logLikelihoodVariance logical\textgreater} forceAcceptance\arginout) Evaluate the likelihood.

void functionChanged() Respond to possible changes in the likelihood function.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void restore(\textcolor{red}{\textless double precision\textgreater} simulationState\argin,\textcolor{r double precision\textgreater} logLikelihood\argin) Log a report on convergence state to the given file unit.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

logical willEvaluate(\textcolor{red}{\textless class((posteriorSampleStateClass))\textgreater} simulationState\arginout,\textcolor{red}{\textless type((modelParameterList))\textgreater} modelParameters_\argin,\textcolor{red}{\textless class((posteriorSampleConvergenceClass))\textgreat simulationConvergence\arginout,\textcolor{red}{\textless double precision\textgreater} temperature\argin,\textcolor{red}{\textless double precision\textgreater} logLikelihoodCurrent\argi double precision\textgreater} logPriorCurrent\argin,\textcolor{red}{\textless double precision\textgreater} logPriorProposed\argin) Returns true if the likelihood will be evaluated for this state.

posteriorSampleLikelihoodGaussianRegression

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((posteriorSampleLikelihoodClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision evaluate(\textcolor{red}{\textless class((posteriorSampleStateClass))\textgreater}
simulationState\arginout,\textcolor{red}{\textless type((modelParameterList))\textgreater}
modelParametersActive_\argin,\textcolor{red}{\textless type((modelParameterList))\textgreater}
modelParametersInactive_\argin,\textcolor{red}{\textless class((posteriorSampleConvergenceClass))\t
simulationConvergence\arginout,\textcolor{red}{\textless double precision\textgreater}
temperature\argin,\textcolor{red}{\textless double precision\textgreater} logLikelihoodCurrent\argi
double precision\textgreater} logPriorCurrent\argin,\textcolor{red}{\textless double
precision\textgreater} logPriorProposed\argin,\textcolor{red}{\textless real\textgreater}
timeEvaluate\arginout,\textcolor{red}{\textless double precision\textgreater} logLikelihoodVariance
logical\textgreater} forceAcceptance\arginout) Evaluate the likelihood.

void functionChanged() Respond to possible changes in the likelihood function.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void restore(\textcolor{red}{\textless double precision\textgreater} simulationState\argin,\textcolor{r
double precision\textgreater} logLikelihood\argin) Log a report on convergence state to the given file unit.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

logical willEvaluate(\textcolor{red}{\textless class((posteriorSampleStateClass))\textgreater} simulationState\arginout,\textcolor{red}{\textless type((modelParameterList))\textgreater} modelParameters_\argin,\textcolor{red}{\textless class((posteriorSampleConvergenceClass))\textgreat simulationConvergence\arginout,\textcolor{red}{\textless double precision\textgreater} temperature\argin,\textcolor{red}{\textless double precision\textgreater} logLikelihoodCurrent\argi double precision\textgreater} logPriorCurrent\argin,\textcolor{red}{\textless double precision\textgreater} logPriorProposed\argin) Returns true if the likelihood will be evaluated for this state.

posteriorSampleLikelihoodHaloMassFunction

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((posteriorSampleLikelihoodClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision evaluate(\textcolor{red}{\textless class((posteriorSampleStateClass))\textgreater} simulationState\arginout,\textcolor{red}{\textless type((modelParameterList))\textgreater} modelParametersActive_\argin,\textcolor{red}{\textless type((modelParameterList))\textgreater} modelParametersInactive_\argin,\textcolor{red}{\textless class((posteriorSampleConvergenceClass))\t simulationConvergence\arginout,\textcolor{red}{\textless double precision\textgreater} temperature\argin,\textcolor{red}{\textless double precision\textgreater} logLikelihoodCurrent\argi double precision\textgreater} logPriorCurrent\argin,\textcolor{red}{\textless double precision\textgreater} logPriorProposed\argin,\textcolor{red}{\textless real\textgreater} timeEvaluate\arginout,\textcolor{red}{\textless double precision\textgreater} logLikelihoodVariance logical\textgreater} forceAcceptance\arginout) Evaluate the likelihood.

void functionChanged() Respond to possible changes in the likelihood function.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void restore(\textcolor{red}{\textless double precision\textgreater} simulationState\argin,\textcolor{red} double precision\textgreater} logLikelihood\argin) Log a report on convergence state to the given file unit.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

logical willEvaluate(\textcolor{red}{\textless class((posteriorSampleStateClass))\textgreater} simulationState\arginout,\textcolor{red}{\textless type((modelParameterList))\textgreater} modelParameters_\argin,\textcolor{red}{\textless class((posteriorSampleConvergenceClass))\textgreat simulationConvergence\arginout,\textcolor{red}{\textless double precision\textgreater} temperature\argin,\textcolor{red}{\textless double precision\textgreater} logLikelihoodCurrent\argi double precision\textgreater} logPriorCurrent\argin,\textcolor{red}{\textless double precision\textgreater} logPriorProposed\argin) Returns true if the likelihood will be evaluated for this state.

posteriorSampleLikelihoodIndependentLikelihoods

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((posteriorSampleLikelihoodClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision evaluate(\textcolor{red}{\textless class((posteriorSampleStateClass))\textgreater} simulationState\arginout,\textcolor{red}{\textless type((modelParameterList))\textgreater} modelParametersActive_\argin,\textcolor{red}{\textless type((modelParameterList))\textgreater} modelParametersInactive_\argin,\textcolor{red}{\textless class((posteriorSampleConvergenceClass))\t simulationConvergence\arginout,\textcolor{red}{\textless double precision\textgreater} temperature\argin,\textcolor{red}{\textless double precision\textgreater} logLikelihoodCurrent\argi double precision\textgreater} logPriorCurrent\argin,\textcolor{red}{\textless double precision\textgreater} logPriorProposed\argin,\textcolor{red}{\textless real\textgreater} timeEvaluate\arginout,\textcolor{red}{\textless double precision\textgreater} logLikelihoodVariance logical\textgreater} forceAcceptance\arginout) Evaluate the likelihood.

void functionChanged() Respond to possible changes in the likelihood function.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void restore(\textcolor{red}{\textless double precision\textgreater} simulationState\argin,\textcolor{r
double precision\textgreater} logLikelihood\argin) Log a report on convergence state to the given file unit.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

logical willEvaluate(\textcolor{red}{\textless class((posteriorSampleStateClass))\textgreater}
simulationState\arginout,\textcolor{red}{\textless type((modelParameterList))\textgreater}
modelParameters_\argin,\textcolor{red}{\textless class((posteriorSampleConvergenceClass))\textgreat
simulationConvergence\arginout,\textcolor{red}{\textless double precision\textgreater}
temperature\argin,\textcolor{red}{\textless double precision\textgreater} logLikelihoodCurrent\argi
double precision\textgreater} logPriorCurrent\argin,\textcolor{red}{\textless double
precision\textgreater} logPriorProposed\argin) Returns true if the likelihood will be evaluated for this state.

posteriorSampleLikelihoodIndpndntLklhdsSqntl

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((posteriorSampleLikelihoodClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision evaluate(\textcolor{red}{\textless class((posteriorSampleStateClass))\textgreater}
simulationState\arginout,\textcolor{red}{\textless type((modelParameterList))\textgreater}
modelParametersActive_\argin,\textcolor{red}{\textless type((modelParameterList))\textgreater}
modelParametersInactive_\argin,\textcolor{red}{\textless class((posteriorSampleConvergenceClass))\t
simulationConvergence\arginout,\textcolor{red}{\textless double precision\textgreater}
temperature\argin,\textcolor{red}{\textless double precision\textgreater} logLikelihoodCurrent\argi
double precision\textgreater} logPriorCurrent\argin,\textcolor{red}{\textless double
precision\textgreater} logPriorProposed\argin,\textcolor{red}{\textless real\textgreater}
timeEvaluate\arginout,\textcolor{red}{\textless double precision\textgreater} logLikelihoodVariance
logical\textgreater} forceAcceptance\arginout) Evaluate the likelihood.

void functionChanged() Respond to possible changes in the likelihood function.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void restore(\textcolor{red}{\textless double precision\textgreater} simulationState\argin,\textcolor{r
double precision\textgreater} logLikelihood\argin) Log a report on convergence state to the given file unit.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

logical willEvaluate(\textcolor{red}{\textless class((posteriorSampleStateClass))\textgreater}
simulationState\arginout,\textcolor{red}{\textless type((modelParameterList))\textgreater}
modelParameters_\argin,\textcolor{red}{\textless class((posteriorSampleConvergenceClass))\textgreat
simulationConvergence\arginout,\textcolor{red}{\textless double precision\textgreater}
temperature\argin,\textcolor{red}{\textless double precision\textgreater} logLikelihoodCurrent\argi
double precision\textgreater} logPriorCurrent\argin,\textcolor{red}{\textless double
precision\textgreater} logPriorProposed\argin) Returns true if the likelihood will be evaluated for this state.

posteriorSampleLikelihoodMassFunction

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((posteriorSampleLikelihoodClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision evaluate(\textcolor{red}{\textless class((posteriorSampleStateClass))\textgreater}
simulationState\arginout,\textcolor{red}{\textless type((modelParameterList))\textgreater}
modelParametersActive_\argin,\textcolor{red}{\textless type((modelParameterList))\textgreater}
modelParametersInactive_\argin,\textcolor{red}{\textless class((posteriorSampleConvergenceClass))\t

simulationConvergence\arginout,\textcolor{red}{\textless double precision\textgreater} temperature\argin,\textcolor{red}{\textless double precision\textgreater} logLikelihoodCurrent\argi double precision\textgreater} logPriorCurrent\argin,\textcolor{red}{\textless double precision\textgreater} logPriorProposed\argin,\textcolor{red}{\textless real\textgreater} timeEvaluate\arginout,\textcolor{red}{\textless double precision\textgreater} logLikelihoodVariance logical\textgreater} forceAcceptance\arginout) Evaluate the likelihood.

void functionChanged() Respond to possible changes in the likelihood function.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void restore(\textcolor{red}{\textless double precision\textgreater} simulationState\argin,\textcolor{r double precision\textgreater} logLikelihood\argin) Log a report on convergence state to the given file unit.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

logical willEvaluate(\textcolor{red}{\textless class((posteriorSampleStateClass))\textgreater} simulationState\arginout,\textcolor{red}{\textless type((modelParameterList))\textgreater} modelParameters_\argin,\textcolor{red}{\textless class((posteriorSampleConvergenceClass))\textgreat simulationConvergence\arginout,\textcolor{red}{\textless double precision\textgreater} temperature\argin,\textcolor{red}{\textless double precision\textgreater} logLikelihoodCurrent\argi double precision\textgreater} logPriorCurrent\argin,\textcolor{red}{\textless double precision\textgreater} logPriorProposed\argin) Returns true if the likelihood will be evaluated for this state.

posteriorSampleLikelihoodMltiVrtNormalStochastic

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((posteriorSampleLikelihoodClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision evaluate(\textcolor{red}{\textless class((posteriorSampleStateClass))\textgreater} simulationState\arginout,\textcolor{red}{\textless type((modelParameterList))\textgreater} modelParametersActive_\argin,\textcolor{red}{\textless type((modelParameterList))\textgreater} modelParametersInactive_\argin,\textcolor{red}{\textless class((posteriorSampleConvergenceClass))\t simulationConvergence\arginout,\textcolor{red}{\textless double precision\textgreater} temperature\argin,\textcolor{red}{\textless double precision\textgreater} logLikelihoodCurrent\argi double precision\textgreater} logPriorCurrent\argin,\textcolor{red}{\textless double precision\textgreater} logPriorProposed\argin,\textcolor{red}{\textless real\textgreater} timeEvaluate\arginout,\textcolor{red}{\textless double precision\textgreater} logLikelihoodVariance logical\textgreater} forceAcceptance\arginout) Evaluate the likelihood.

void functionChanged() Respond to possible changes in the likelihood function.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void restore(\textcolor{red}{\textless double precision\textgreater} simulationState\argin,\textcolor{r double precision\textgreater} logLikelihood\argin) Log a report on convergence state to the given file unit.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

logical willEvaluate(\textcolor{red}{\textless class((posteriorSampleStateClass))\textgreater} simulationState\arginout,\textcolor{red}{\textless type((modelParameterList))\textgreater} modelParameters_\argin,\textcolor{red}{\textless class((posteriorSampleConvergenceClass))\textgreat simulationConvergence\arginout,\textcolor{red}{\textless double precision\textgreater} temperature\argin,\textcolor{red}{\textless double precision\textgreater} logLikelihoodCurrent\argi double precision\textgreater} logPriorCurrent\argin,\textcolor{red}{\textless double precision\textgreater} logPriorProposed\argin) Returns true if the likelihood will be evaluated for this state.

posteriorSampleLikelihoodMultivariateNormal

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((posteriorSampleLikelihoodClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision evaluate(\textcolor{red}{\textless class((posteriorSampleStateClass))\textgreater} simulationState\arginout,\textcolor{red}{\textless type((modelParameterList))\textgreater} modelParametersActive_\argin,\textcolor{red}{\textless type((modelParameterList))\textgreater} modelParametersInactive_\argin,\textcolor{red}{\textless class((posteriorSampleConvergenceClass))\t simulationConvergence\arginout,\textcolor{red}{\textless double precision\textgreater} temperature\argin,\textcolor{red}{\textless double precision\textgreater} logLikelihoodCurrent\argi double precision\textgreater} logPriorCurrent\argin,\textcolor{red}{\textless double precision\textgreater} logPriorProposed\argin,\textcolor{red}{\textless real\textgreater} timeEvaluate\arginout,\textcolor{red}{\textless double precision\textgreater} logLikelihoodVariance logical\textgreater} forceAcceptance\arginout)` Evaluate the likelihood.

`void functionChanged()` Respond to possible changes in the likelihood function.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void restore(\textcolor{red}{\textless double precision\textgreater} simulationState\argin,\textcolor{r double precision\textgreater} logLikelihood\argin)` Log a report on convergence state to the given file unit.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

logical willEvaluate(\textcolor{red}{\textless class((posteriorSampleStateClass))\textgreater} simulationState\arginout,\textcolor{red}{\textless type((modelParameterList))\textgreater} modelParameters_\argin,\textcolor{red}{\textless class((posteriorSampleConvergenceClass))\textgreat simulationConvergence\arginout,\textcolor{red}{\textless double precision\textgreater} temperature\argin,\textcolor{red}{\textless double precision\textgreater} logLikelihoodCurrent\argi double precision\textgreater} logPriorCurrent\argin,\textcolor{red}{\textless double precision\textgreater} logPriorProposed\argin) Returns true if the likelihood will be evaluated for this state.

posteriorSampleLikelihoodPosteriorAsPrior

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((posteriorSampleLikelihoodClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision evaluate(\textcolor{red}{\textless class((posteriorSampleStateClass))\textgreater} simulationState\arginout,\textcolor{red}{\textless type((modelParameterList))\textgreater} modelParametersActive_\argin,\textcolor{red}{\textless type((modelParameterList))\textgreater} modelParametersInactive_\argin,\textcolor{red}{\textless class((posteriorSampleConvergenceClass))\t simulationConvergence\arginout,\textcolor{red}{\textless double precision\textgreater} temperature\argin,\textcolor{red}{\textless double precision\textgreater} logLikelihoodCurrent\argi double precision\textgreater} logPriorCurrent\argin,\textcolor{red}{\textless double precision\textgreater} logPriorProposed\argin,\textcolor{red}{\textless real\textgreater} timeEvaluate\arginout,\textcolor{red}{\textless double precision\textgreater} logLikelihoodVariance logical\textgreater} forceAcceptance\arginout) Evaluate the likelihood.

void functionChanged() Respond to possible changes in the likelihood function.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<void> initialize(<type(mappingList)[:]> modelParametersActive_→) Initialize the posterior-as-prior likelihood.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void restore(\textcolor{red}{\textless double precision\textgreater} simulationState\argin,\textcolor{r
double precision\textgreater} logLikelihood\argin) Log a report on convergence state to the given file unit.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

logical willEvaluate(\textcolor{red}{\textless class((posteriorSampleStateClass))\textgreater}
simulationState\arginout,\textcolor{red}{\textless type((modelParameterList))\textgreater}
modelParameters_\argin,\textcolor{red}{\textless class((posteriorSampleConvergenceClass))\textgreat
simulationConvergence\arginout,\textcolor{red}{\textless double precision\textgreater}
temperature\argin,\textcolor{red}{\textless double precision\textgreater} logLikelihoodCurrent\argi
double precision\textgreater} logPriorCurrent\argin,\textcolor{red}{\textless double
precision\textgreater} logPriorProposed\argin) Returns true if the likelihood will be evaluated for this state.

posteriorSampleLikelihoodPrjctdCorrelationFunction

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((posteriorSampleLikelihoodClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision evaluate(\textcolor{red}{\textless class((posteriorSampleStateClass))\textgreater}
simulationState\arginout,\textcolor{red}{\textless type((modelParameterList))\textgreater}
modelParametersActive_\argin,\textcolor{red}{\textless type((modelParameterList))\textgreater}
modelParametersInactive_\argin,\textcolor{red}{\textless class((posteriorSampleConvergenceClass))\t
simulationConvergence\arginout,\textcolor{red}{\textless double precision\textgreater}
temperature\argin,\textcolor{red}{\textless double precision\textgreater} logLikelihoodCurrent\argi
double precision\textgreater} logPriorCurrent\argin,\textcolor{red}{\textless double
precision\textgreater} logPriorProposed\argin,\textcolor{red}{\textless real\textgreater}
timeEvaluate\arginout,\textcolor{red}{\textless double precision\textgreater} logLikelihoodVariance
logical\textgreater} forceAcceptance\arginout) Evaluate the likelihood.

void functionChanged() Respond to possible changes in the likelihood function.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void restore(\textcolor{red}{\textless double precision\textgreater} simulationState\argin,\textcolor{r double precision\textgreater} logLikelihood\argin) Log a report on convergence state to the given file unit.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

logical willEvaluate(\textcolor{red}{\textless class((posteriorSampleStateClass))\textgreater} simulationState\arginout,\textcolor{red}{\textless type((modelParameterList))\textgreater} modelParameters_\argin,\textcolor{red}{\textless class((posteriorSampleConvergenceClass))\textgreat simulationConvergence\arginout,\textcolor{red}{\textless double precision\textgreater} temperature\argin,\textcolor{red}{\textless double precision\textgreater} logLikelihoodCurrent\argi double precision\textgreater} logPriorCurrent\argin,\textcolor{red}{\textless double precision\textgreater} logPriorProposed\argin) Returns true if the likelihood will be evaluated for this state.

posteriorSampleLikelihoodSEDFit

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((posteriorSampleLikelihoodClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision evaluate(\textcolor{red}{\textless class((posteriorSampleStateClass))\textgreater} simulationState\arginout,\textcolor{red}{\textless type((modelParameterList))\textgreater} modelParametersActive_\argin,\textcolor{red}{\textless type((modelParameterList))\textgreater} modelParametersInactive_\argin,\textcolor{red}{\textless class((posteriorSampleConvergenceClass))\t simulationConvergence\arginout,\textcolor{red}{\textless double precision\textgreater} temperature\argin,\textcolor{red}{\textless double precision\textgreater} logLikelihoodCurrent\argi double precision\textgreater} logPriorCurrent\argin,\textcolor{red}{\textless double precision\textgreater} logPriorProposed\argin,\textcolor{red}{\textless real\textgreater} timeEvaluate\arginout,\textcolor{red}{\textless double precision\textgreater} logLikelihoodVariance logical\textgreater} forceAcceptance\arginout) Evaluate the likelihood.

`void functionChanged()` Respond to possible changes in the likelihood function.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void restore(\textcolor{red}{\textless double precision\textgreater} simulationState\argin,\textcolor{r` `double precision\textgreater} logLikelihood\argin)` Log a report on convergence state to the given file unit.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl` `type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-` `size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles` `type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-` `size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`logical willEvaluate(\textcolor{red}{\textless class((posteriorSampleStateClass ))\textgreater}` `simulationState\arginout,\textcolor{red}{\textless type((modelParameterList ))\textgreater}` `modelParameters_\argin,\textcolor{red}{\textless class((posteriorSampleConvergenceClass))\textgreat` `simulationConvergence\arginout,\textcolor{red}{\textless double precision\textgreater}` `temperature\argin,\textcolor{red}{\textless double precision\textgreater} logLikelihoodCurrent\argi` `double precision\textgreater} logPriorCurrent\argin,\textcolor{red}{\textless double` `precision\textgreater} logPriorProposed\argin)` Returns true if the likelihood will be evaluated for this state.

posteriorSampleLikelihoodSpinDistribution

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\` `character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((posteriorSampleLikelihoodClass))\textgreater}` `destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex` `logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

double precision evaluate(\textcolor{red}{\textless class((posteriorSampleStateClass))\textgreater} simulationState\arginout,\textcolor{red}{\textless type((modelParameterList))\textgreater} modelParametersActive_\argin,\textcolor{red}{\textless type((modelParameterList))\textgreater} modelParametersInactive_\argin,\textcolor{red}{\textless class((posteriorSampleConvergenceClass))\t simulationConvergence\arginout,\textcolor{red}{\textless double precision\textgreater} temperature\argin,\textcolor{red}{\textless double precision\textgreater} logLikelihoodCurrent\argi double precision\textgreater} logPriorCurrent\argin,\textcolor{red}{\textless double precision\textgreater} logPriorProposed\argin,\textcolor{red}{\textless real\textgreater} timeEvaluate\arginout,\textcolor{red}{\textless double precision\textgreater} logLikelihoodVariance logical\textgreater} forceAcceptance\arginout) Evaluate the likelihood.

void functionChanged() Respond to possible changes in the likelihood function.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void restore(\textcolor{red}{\textless double precision\textgreater} simulationState\argin,\textcolor{r double precision\textgreater} logLikelihood\argin) Log a report on convergence state to the given file unit.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

logical willEvaluate(\textcolor{red}{\textless class((posteriorSampleStateClass))\textgreater} simulationState\arginout,\textcolor{red}{\textless type((modelParameterList))\textgreater} modelParameters_\argin,\textcolor{red}{\textless class((posteriorSampleConvergenceClass))\textgreat simulationConvergence\arginout,\textcolor{red}{\textless double precision\textgreater} temperature\argin,\textcolor{red}{\textless double precision\textgreater} logLikelihoodCurrent\argi double precision\textgreater} logPriorCurrent\argin,\textcolor{red}{\textless double precision\textgreater} logPriorProposed\argin) Returns true if the likelihood will be evaluated for this state.

posteriorSampleSimulationAnnealedDffrntlEvltn

<logical> acceptProposal(<double> logPosterior→, <double> logPosteriorProposed→, <type(pseudoRandom)> randomNumberGenerator↔) Return true if the proposed state should be accepted.

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

<integer> chainSelect(<type(pseudoRandom)> randomNumberGenertor↔, <integer(:)> [blockedChains]→) Select a chain.

void deepCopy(\textcolor{red}{\textless class((posteriorSampleSimulationClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<void> initialize(<integer> temperatureLevelCount→, <double> temperatureMaximum) Initialize the object.

<logical> isDefault() Return true if this is the default object of this class.

<integer> level() Return the current tempering level.

<logical> logging() Return true if the simulator is currently logging state.

type(varying_string) objectType() Return the type of the object.

<double> posterior(<class(posteriorSampleStateClass)> posteriorSampleState_→) Return the log of posterior probability for the given posteriorSampleState.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void simulate() Perform the simulation.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

<double> stepSize() Return the step size parameter, γ, for the differential evolution proposal vector.

<double> temperature() Return the current temperature.

<void> update(<double(:)> stateVector→) Update the simulator to the new stateVector after a step.

posteriorSampleSimulationClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((posteriorSampleSimulationClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void simulate() Perform the simulation.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

posteriorSampleSimulationDifferentialEvolution

<logical> acceptProposal(<double> logPosterior→, <double> logPosteriorProposed→, <type(pseudoRandom)> randomNumberGenerator↔) Return true if the proposed state should be accepted.

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

<integer> chainSelect(<type(pseudoRandom)> randomNumberGenertor↔, <integer(:)> [blockedChains]→) Select a chain.

void deepCopy(\textcolor{red}{\textless class((posteriorSampleSimulationClass))\textgreater} destination\arginout) Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`<logical> logging()` Return true if the simulator is currently logging state.

`type(varying_string) objectType()` Return the type of the object.

`<double> posterior(<class(posteriorSampleStateClass)> posteriorSampleState_→)` Return the log of posterior probability for the given `posteriorSampleState`.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void simulate()` Perform the simulation.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`<double> stepSize()` Return the step size parameter, γ, for the differential evolution proposal vector.

`<double> temperature()` Return the current temperature.

`<void> update(<double(:)> stateVector→)` Update the simulator to the new `stateVector` after a step.

posteriorSampleSimulationParticleSwarm

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((posteriorSampleSimulationClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<double> posterior(<class(posteriorSampleStateClass)> simulationState→) Return the log of posterior probability for the given simulationState.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void simulate() Perform the simulation.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

posteriorSampleSimulationStochasticDffrntlEvltn

<logical> acceptProposal(<double> logPosterior→, <double> logPosteriorProposed→, <type(pseudoRandom)> randomNumberGenerator↔) Return true if the proposed state should be accepted.

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

<integer> chainSelect(<type(pseudoRandom)> randomNumberGenertor↔, <integer(:)> [blockedChains]→) Select a chain.

void deepCopy(\textcolor{red}{\textless class((posteriorSampleSimulationClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<void> initialize(<integer> temperatureLevelCount→, <double> temperatureMaximum) Initialize the object.

<logical> isDefault() Return true if this is the default object of this class.

<logical> logging() Return true if the simulator is currently logging state.

type(varying_string) objectType() Return the type of the object.

<double> posterior(<class(posteriorSampleStateClass)> posteriorSampleState_→) Return the log of posterior probability for the given posteriorSampleState.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void simulate() Perform the simulation.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

<double> stepSize() Return the step size parameter, γ, for the differential evolution proposal vector.

<double> temperature() Return the current temperature.

<void> update(<double(:)> stateVector→) Update the simulator to the new stateVector after a step.

posteriorSampleSimulationTemperedDffrntlEvltn

<logical> acceptProposal(<double> logPosterior→, <double> logPosteriorProposed→, <type(pseudoRandom)> randomNumberGenerator↔) Return true if the proposed state should be accepted.

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

<integer> chainSelect(<type(pseudoRandom)> randomNumberGenertor↔, <integer(:)> [blockedChains]→) Select a chain.

void deepCopy(\textcolor{red}{\textless class((posteriorSampleSimulationClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<void> initialize(<class(posteriorSampleDffrntlEvltnPrpslSzTmpExpClass)> posteriorSampleDffrntlEvltnPrp →) Return the current tempering level.

<logical> isDefault() Return true if this is the default object of this class.

<integer> level() Return the current tempering level.

<logical> logging() Return true if the simulator is currently logging state.

type(varying_string) objectType() Return the type of the object.

<double> posterior(<class(posteriorSampleStateClass)> posteriorSampleState_→) Return the log of posterior probability for the given posteriorSampleState.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void simulate() Perform the simulation.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

<double> stepSize() Return the step size parameter, γ, for the differential evolution proposal vector.

<double> temperature() Return the current temperature.

<void> update(<double(:)> stateVector→) Update the simulator to the new stateVector after a step.

posteriorSampleStateClass

double precision acceptanceRate() Return the state acceptance rate.

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

integer chainIndex() Return the index of the chain associated with this state.

void chainIndexSet(\textcolor{red}{\textless integer\textgreater} chainIndex\argin) Set the index of the chain associated with this state.

integer count() Returns the number of steps in the current state.

void deepCopy(\textcolor{red}{\textless class((posteriorSampleStateClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

`integer dimension()` Returns the dimension of the state.

`double precision, dimension(self%parameterCount) get()` Get the current state vector.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision, dimension(self%parameterCount) mean()` Return the mean state.

`type(varying_string) objectType()` Return the type of the object.

`void parameterCountSet(\textcolor{red}{\textless integer\textgreater} parameterCount\argin)` Set the number of parameters in this state.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void reset()` Reset the state object.

`void restore(\textcolor{red}{\textless double precision\textgreater} stateVector\argin,\textcolor{red}{` `logical\textgreater} first\argin)` Restore the state, one step at a time.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl` `type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-` `size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles` `type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-` `size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`void update(\textcolor{red}{\textless double precision\textgreater} stateNew\argin,\textcolor{red}{\tex` `logical\textgreater} logState\argin,\textcolor{red}{\textless logical\textgreater}` `isConverged\argin,\textcolor{red}{\textless logical\textgreater} outlierMask\argin)` Update the state vector.

`double precision, dimension(self%parameterCount) variance()` Return the variance in the state vector.

posteriorSampleStateCorrelation

`double precision acceptanceRate()` Return the state acceptance rate.

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\` `character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`integer chainIndex()` Return the index of the chain associated with this state.

`void chainIndexSet(\textcolor{red}{\textless integer\textgreater} chainIndex\argin)` Set the index of the chain associated with this state.

`<integer> correlationLength()` Return the current correlation length in the chains.

`<integer> correlationLengthCompute(<logical(:)> [outlierMask]→)` Compute correlation lengths in the chains.

`integer count()` Returns the number of steps in the current state.

`void deepCopy(\textcolor{red}{\textless class((posteriorSampleStateClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`integer dimension()` Returns the dimension of the state.

`double precision, dimension(self%parameterCount) get()` Get the current state vector.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision, dimension(self%parameterCount) mean()` Return the mean state.

`type(varying_string) objectType()` Return the type of the object.

`void parameterCountSet(\textcolor{red}{\textless integer\textgreater} parameterCount\argin)` Set the number of parameters in this state.

`<integer> postConvergenceCorrelationCount()` Return the number of post-convergence correlation lengths that have accrued.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void reset()` Reset the state object.

`void restore(\textcolor{red}{\textless double precision\textgreater} stateVector\argin,\textcolor{red}{ logical\textgreater} first\argin)` Restore the state, one step at a time.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`void update(\textcolor{red}{\textless double precision\textgreater} stateNew\argin,\textcolor{red}{\tex logical\textgreater} logState\argin,\textcolor{red}{\textless logical\textgreater} isConverged\argin,\textcolor{red}{\textless logical\textgreater} outlierMask\argin)` Update the state vector.

`double precision, dimension(self%parameterCount) variance()` Return the variance in the state vector.

posteriorSampleStateHistory

`double precision acceptanceRate()` Return the state acceptance rate.

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`integer chainIndex()` Return the index of the chain associated with this state.

`void chainIndexSet(\textcolor{red}{\textless integer\textgreater} chainIndex\argin)` Set the index of the chain associated with this state.

`integer count()` Returns the number of steps in the current state.

`void deepCopy(\textcolor{red}{\textless class((posteriorSampleStateClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`integer dimension()` Returns the dimension of the state.

`double precision, dimension(self%parameterCount) get()` Get the current state vector.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision, dimension(self%parameterCount) mean()` Return the mean state.

`type(varying_string) objectType()` Return the type of the object.

`void parameterCountSet(\textcolor{red}{\textless integer\textgreater} parameterCount\argin)` Set the number of parameters in this state.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void reset()` Reset the state object.

`void restore(\textcolor{red}{\textless double precision\textgreater} stateVector\argin,\textcolor{red}{\textless logical\textgreater} first\argin)` Restore the state, one step at a time.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`void update(\textcolor{red}{\textless double precision\textgreater} stateNew\argin,\textcolor{red}{\textless logical\textgreater} logState\argin,\textcolor{red}{\textless logical\textgreater} isConverged\argin,\textcolor{red}{\textless logical\textgreater} outlierMask\argin)` Update the state vector.

`double precision, dimension(self%parameterCount) variance()` Return the variance in the state vector.

posteriorSampleStateInitializeClass

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\textcolor{red}{\textless character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((posteriorSampleStateInitializeClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\textcolor{red}{\textless logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDigest\argin)` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`void initialize(\textcolor{red}{\textless class((posteriorSampleStateClass))\textgreater} simulationState\arginout,\textcolor{red}{\textless type((modelParameterList))\textgreater} modelParameters_\argin,\textcolor{red}{\textless class((posteriorSampleLikelihoodClass))\textgreater} modelLikelihood\arginout,\textcolor{red}{\textless double precision\textgreater} timeEvaluatePrevious\argin,\textcolor{red}{\textless double precision\textgreater} logLikelihood\argout,\textcolor{red}{\textless double precision\textgreater} logPosterior\argout)` Initialize the state of the posterior sampler.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

posteriorSampleStateInitializeLatinHypercube

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((posteriorSampleStateInitializeClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`void initialize(\textcolor{red}{\textless class((posteriorSampleStateClass))\textgreater} simulationState\arginout,\textcolor{red}{\textless type((modelParameterList))\textgreater} modelParameters_\argin,\textcolor{red}{\textless class((posteriorSampleLikelihoodClass))\textgreate modelLikelihood\arginout,\textcolor{red}{\textless double precision\textgreater} timeEvaluatePrevio double precision\textgreater} logLikelihood\argout,\textcolor{red}{\textless double precision\textgreater} logPosterior\argout)` Initialize the state of the posterior sampler.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

posteriorSampleStateInitializePriorRandom

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((posteriorSampleStateInitializeClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

void initialize(\textcolor{red}{\textless class((posteriorSampleStateClass))\textgreater}
simulationState\arginout,\textcolor{red}{\textless type((modelParameterList))\textgreater}
modelParameters_\argin,\textcolor{red}{\textless class((posteriorSampleLikelihoodClass))\textgreate
modelLikelihood\arginout,\textcolor{red}{\textless double precision\textgreater} timeEvaluatePrevio
double precision\textgreater} logLikelihood\argout,\textcolor{red}{\textless double
precision\textgreater} logPosterior\argout) Initialize the state of the posterior sampler.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

posteriorSampleStateInitializeResume

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((posteriorSampleStateInitializeClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`void initialize(\textcolor{red}{\textless class((posteriorSampleStateClass))\textgreater} simulationState\arginout,\textcolor{red}{\textless type((modelParameterList))\textgreater} modelParameters_\argin,\textcolor{red}{\textless class((posteriorSampleLikelihoodClass))\textgreate modelLikelihood\arginout,\textcolor{red}{\textless double precision\textgreater} timeEvaluatePrevio double precision\textgreater} logLikelihood\argout,\textcolor{red}{\textless double precision\textgreater} logPosterior\argout)` Initialize the state of the posterior sampler.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

posteriorSampleStateInitializeSwitched

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((posteriorSampleStateInitializeClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`void initialize(\textcolor{red}{\textless class((posteriorSampleStateClass))\textgreater} simulationState\arginout,\textcolor{red}{\textless type((modelParameterList))\textgreater} modelParameters_\argin,\textcolor{red}{\textless class((posteriorSampleLikelihoodClass))\textgreate modelLikelihood\arginout,\textcolor{red}{\textless double precision\textgreater} timeEvaluatePrevio`

double precision\textgreater} logLikelihood\argout,\textcolor{red}{\textless double precision\textgreater} logPosterior\argout) Initialize the state of the posterior sampler.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

posteriorSampleStateSimple

double precision acceptanceRate() Return the state acceptance rate.

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

integer chainIndex() Return the index of the chain associated with this state.

void chainIndexSet(\textcolor{red}{\textless integer\textgreater} chainIndex\argin) Set the index of the chain associated with this state.

integer count() Returns the number of steps in the current state.

void deepCopy(\textcolor{red}{\textless class((posteriorSampleStateClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

integer dimension() Returns the dimension of the state.

double precision, dimension(self%parameterCount) get() Get the current state vector.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision, dimension(self%parameterCount) mean() Return the mean state.

type(varying_string) objectType() Return the type of the object.

void parameterCountSet(\textcolor{red}{\textless integer\textgreater} parameterCount\argin) Set the number of parameters in this state.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void reset() Reset the state object.

void restore(\textcolor{red}{\textless double precision\textgreater} stateVector\argin,\textcolor{red}{\ logical\textgreater} first\argin) Restore the state, one step at a time.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

void update(\textcolor{red}{\textless double precision\textgreater} stateNew\argin,\textcolor{red}{\tex logical\textgreater} logState\argin,\textcolor{red}{\textless logical\textgreater} isConverged\argin,\textcolor{red}{\textless logical\textgreater} outlierMask\argin) Update the state vector.

double precision, dimension(self%parameterCount) variance() Return the variance in the state vector.

posteriorSampleStoppingCriterionClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((posteriorSampleStoppingCriterionClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

logical stop(\textcolor{red}{\textless class((posteriorSampleStateClass))\textgreater} simulationState\arginout) Returns true if the posterior sampling should stop.

posteriorSampleStoppingCriterionCorrelationLength

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((posteriorSampleStoppingCriterionClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

logical stop(\textcolor{red}{\textless class((posteriorSampleStateClass))\textgreater} simulationState\arginout) Returns true if the posterior sampling should stop.

posteriorSampleStoppingCriterionNever

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((posteriorSampleStoppingCriterionClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`logical stop(\textcolor{red}{\textless class((posteriorSampleStateClass))\textgreater} simulationState\arginout)` Returns true if the posterior sampling should stop.

posteriorSampleStoppingCriterionStepCount

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((posteriorSampleStoppingCriterionClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

logical stop(\textcolor{red}{\textless class((posteriorSampleStateClass))\textgreater}
simulationState\arginout) Returns true if the posterior sampling should stop.

powerSpectrumClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((powerSpectrumClass))\textgreater} destination\arginout)
Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision power(\textcolor{red}{\textless double precision\textgreater} wavenumber\argin)
Return the linear power spectrum for $k =$wavenumber [Mpc^{-1}].

double precision powerDimensionless(\textcolor{red}{\textless double precision\textgreater}
wavenumber\argin) Return the dimensionless linear power spectrum for $k =$wavenumber [Mpc^{-1}].

double precision powerLogarithmicDerivative(\textcolor{red}{\textless double precision\textgreater}
wavenumber\argin) Return the logarithmic derivative of the power spectrum, $\mathrm{d}\ln P(k)/\mathrm{d}\ln k$, for $k =$wavenumber [Mpc^{-1}].

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

powerSpectrumNonlinearClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((powerSpectrumNonlinearClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision value(\textcolor{red}{\textless double precision\textgreater} wavenumber\argin,\textco double precision\textgreater} time\argin) Return the nonlinear power spectrum for $k =$wavenumber [Mpc^{-1}] at cosmic time $t =$time [Gyr].

powerSpectrumNonlinearCosmicEmu

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((powerSpectrumNonlinearClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision value(\textcolor{red}{\textless double precision\textgreater} wavenumber\argin,\textco double precision\textgreater} time\argin)` Return the nonlinear power spectrum for $k =$`wavenumber` [Mpc^{-1}] at cosmic time $t =$`time` [Gyr].

powerSpectrumNonlinearLinear

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((powerSpectrumNonlinearClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision value(\textcolor{red}{\textless double precision\textgreater} wavenumber\argin,\textco
double precision\textgreater} time\argin) Return the nonlinear power spectrum for $k =$wavenumber [Mpc^{-1}] at cosmic time $t =$time [Gyr].

powerSpectrumNonlinearPeacockDodds1996

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((powerSpectrumNonlinearClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision value(\textcolor{red}{\textless double precision\textgreater} wavenumber\argin,\textco double precision\textgreater} time\argin) Return the nonlinear power spectrum for $k =$wavenumber [Mpc^{-1}] at cosmic time $t =$time [Gyr].

powerSpectrumPrimordialClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((powerSpectrumPrimordialClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision logarithmicDerivative(\textcolor{red}{\textless double precision\textgreater} wavenumber\argin) Return the logarithmic derivative with respect to wavenumber of the primordial power spectrum at the given wavenumber (specified in units of Mpc^{-1}.

type(varying_string) objectType() Return the type of the object.

double precision power(\textcolor{red}{\textless double precision\textgreater} wavenumber\argin) Return the (unnormalized) power in the primordial power spectrum at the given wavenumber (specified in units of Mpc^{-1}.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

powerSpectrumPrimordialPowerLaw

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((powerSpectrumPrimordialClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision logarithmicDerivative(\textcolor{red}{\textless double precision\textgreater} wavenumber\argin)` Return the logarithmic derivative with respect to wavenumber of the primordial power spectrum at the given `wavenumber` (specified in units of Mpc^{-1}.

`type(varying_string) objectType()` Return the type of the object.

`double precision power(\textcolor{red}{\textless double precision\textgreater} wavenumber\argin)` Return the (unnormalized) power in the primordial power spectrum at the given `wavenumber` (specified in units of Mpc^{-1}.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

powerSpectrumPrimordialTransferredClass

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((powerSpectrumPrimordialTransferredClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision logarithmicDerivative(\textcolor{red}{\textless double precision\textgreater} wavenumber\argin)` Return the logarithmic derivative with respect to wavenumber of the transferred primordial power spectrum at the given `wavenumber` (specified in units of Mpc^{-1}.

`type(varying_string) objectType()` Return the type of the object.

`double precision power(\textcolor{red}{\textless double precision\textgreater} wavenumber\argin)` Return the (unnormalized) power in the transferred primordial power spectrum at the given `wavenumber` (specified in units of Mpc^{-1}.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

powerSpectrumPrimordialTransferredSimple

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((powerSpectrumPrimordialTransferredClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision logarithmicDerivative(\textcolor{red}{\textless double precision\textgreater} wavenumber\argin)` Return the logarithmic derivative with respect to wavenumber of the transferred primordial power spectrum at the given `wavenumber` (specified in units of Mpc^{-1}.

`type(varying_string) objectType()` Return the type of the object.

`double precision power(\textcolor{red}{\textless double precision\textgreater} wavenumber\argin)` Return the (unnormalized) power in the transferred primordial power spectrum at the given `wavenumber` (specified in units of Mpc^{-1}.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

powerSpectrumStandard

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((powerSpectrumClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`double precision power(\textcolor{red}{\textless double precision\textgreater} wavenumber\argin)` Return the linear power spectrum for $k =$`wavenumber` $[\mathrm{Mpc}^{-1}]$.

`double precision powerDimensionless(\textcolor{red}{\textless double precision\textgreater} wavenumber\argin)` Return the dimensionless linear power spectrum for $k =$`wavenumber` $[\mathrm{Mpc}^{-1}]$.

`double precision powerLogarithmicDerivative(\textcolor{red}{\textless double precision\textgreater} wavenumber\argin)` Return the logarithmic derivative of the power spectrum, $\mathrm{d}\ln P(k)/\mathrm{d}\ln k$, for $k =$`wavenumber` $[\mathrm{Mpc}^{-1}]$.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

powerSpectrumWindowFunctionClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

logical amplitudeIsMassIndependent() Should return true if, and only if, the amplitude of the window function below the maximum wavenumber is independent of the smoothing mass scale.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((powerSpectrumWindowFunctionClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

`double precision value(\textcolor{red}{\textless double precision\textgreater} wavenumber\argin,\textco double precision\textgreater} smoothingMass\argin)` Returns the window function for power spectrum variance computation at the specified `wavenumber` (in Mpc^{-1}) for a given `smoothingMass` (in $M_\odot$).

`double precision wavenumberMaximum(\textcolor{red}{\textless double precision\textgreater} smoothingMass\argin)` Returns the maximum wavenumber for which the window function for power spectrum variance computation is non-zero for a given `smoothingMass` (in $M_\odot$).

powerSpectrumWindowFunctionLagrangianChan2017

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`logical amplitudeIsMassIndependent()` Should return true if, and only if, the amplitude of the window function below the maximum wavenumber is independent of the smoothing mass scale.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((powerSpectrumWindowFunctionClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision value(\textcolor{red}{\textless double precision\textgreater} wavenumber\argin,\textco double precision\textgreater} smoothingMass\argin)` Returns the window function for power spectrum variance computation at the specified `wavenumber` (in Mpc^{-1}) for a given `smoothingMass` (in $M_\odot$).

`double precision wavenumberMaximum(\textcolor{red}{\textless double precision\textgreater} smoothingMass\argin)` Returns the maximum wavenumber for which the window function for power spectrum variance computation is non-zero for a given `smoothingMass` (in $M_\odot$).

powerSpectrumWindowFunctionSharpKSpace

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

logical amplitudeIsMassIndependent() Should return true if, and only if, the amplitude of the window function below the maximum wavenumber is independent of the smoothing mass scale.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((powerSpectrumWindowFunctionClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision value(\textcolor{red}{\textless double precision\textgreater} wavenumber\argin,\textco
double precision\textgreater} smoothingMass\argin) Returns the window function for power spectrum variance computation at the specified wavenumber (in Mpc^{-1}) for a given smoothingMass (in $M_\odot$).

double precision wavenumberMaximum(\textcolor{red}{\textless double precision\textgreater} smoothingMass\argin) Returns the maximum wavenumber for which the window function for power spectrum variance computation is non-zero for a given smoothingMass (in $M_\odot$).

powerSpectrumWindowFunctionSmoothKSpace

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

`logical amplitudeIsMassIndependent()` Should return true if, and only if, the amplitude of the window function below the maximum wavenumber is independent of the smoothing mass scale.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((powerSpectrumWindowFunctionClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision value(\textcolor{red}{\textless double precision\textgreater} wavenumber\argin,\textco double precision\textgreater} smoothingMass\argin)` Returns the window function for power spectrum variance computation at the specified `wavenumber` (in Mpc^{-1}) for a given `smoothingMass` (in $M_\odot$).

`double precision wavenumberMaximum(\textcolor{red}{\textless double precision\textgreater} smoothingMass\argin)` Returns the maximum wavenumber for which the window function for power spectrum variance computation is non-zero for a given `smoothingMass` (in $M_\odot$).

powerSpectrumWindowFunctionTopHat

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`logical amplitudeIsMassIndependent()` Should return true if, and only if, the amplitude of the window function below the maximum wavenumber is independent of the smoothing mass scale.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((powerSpectrumWindowFunctionClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision value(\textcolor{red}{\textless double precision\textgreater} wavenumber\argin,\textco double precision\textgreater} smoothingMass\argin)` Returns the window function for power spectrum variance computation at the specified `wavenumber` (in Mpc^{-1}) for a given `smoothingMass` (in $M_\odot$).

`double precision wavenumberMaximum(\textcolor{red}{\textless double precision\textgreater} smoothingMass\argin)` Returns the maximum wavenumber for which the window function for power spectrum variance computation is non-zero for a given `smoothingMass` (in $M_\odot$).

powerSpectrumWindowFunctionTopHatSharpKHybrid

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`logical amplitudeIsMassIndependent()` Should return true if, and only if, the amplitude of the window function below the maximum wavenumber is independent of the smoothing mass scale.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((powerSpectrumWindowFunctionClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<void> radii(<double> smoothingMass→,<double> radiusTopHat→,<double> radiusKSpaceSharp→) Set the radii of the components of the window function.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision value(\textcolor{red}{\textless double precision\textgreater} wavenumber\argin,\textco double precision\textgreater} smoothingMass\argin) Returns the window function for power spectrum variance computation at the specified wavenumber (in Mpc^{-1}) for a given smoothingMass (in $M_\odot$).

double precision wavenumberMaximum(\textcolor{red}{\textless double precision\textgreater} smoothingMass\argin) Returns the maximum wavenumber for which the window function for power spectrum variance computation is non-zero for a given smoothingMass (in $M_\odot$).

progenitorIterator

<*class(nodeData)> current(<class(nodeData)(:)> nodes→) Return a pointer to the current progenitor.

<void> descendentSet(<class(nodeData)> node→, <class(nodeData)(:)> nodes→) Set the target descendent node and initialize the iterator.

<logical> exist() Return true if any progenitors exist, false otherwise.

<integer(kind=kind_int8)> index(<class(nodeData)(:)> nodes→) Return the index of the current progenitor.

<logical> next(<class(nodeData)(:)> nodes→) Move to the next progenitor. Returns true if the next progenitor exists, false otherwise.

pseudoRandom

<type(pseudoRandom)> clone() Clone a pseudo-random number generator.

<double> normalSample(<double> mean) Return a pseudo-random number drawn from a standard normal distribution.

<integer> poissonSample(<double> mean) Return a pseudo-random number drawn from a Poisson distribution of the given mean.

<void> restore(<integer> stateFile→, <type(fgsl_file)> fgslStateFile→) Restore a pseudo-random number generator state from file.

<void> store(<integer> stateFile→, <type(fgsl_file)> fgslStateFile→) Store a pseudo-random number generator state to file.

<double> uniformSample() Return a pseudo-random number drawn from a uniform distribution on the interval 0 to 1.

radiationFieldBlackBody

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((radiationFieldClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision flux(\textcolor{red}{\textless double precision\textgreater} wavelength\argin,\textcol type((treeNode))\textgreater} node\arginout) Return the flux (in units of ergs cm^2 s^{-1} Hz^{-1} ster^{-1}) of the given radiation field.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

double precision integrateOverCrossSection(\textcolor{red}{\textless double precision\textgreater} wavelengthRange\argin,\textcolor{red}{\textless double precision\textgreater} crossSectionFunction, type((treeNode))\textgreater} node\arginout) Integrates the flux (in units of ergs cm^2 s^{-1} Hz^{-1} ster^{-1}) of the given radiation structure between the wavelengths given in wavelengthRange over a cross section specified by the function crossSectionFunction.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

<double> temperature() Return the temperature of te black-body radiation field.

radiationFieldClass

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((radiationFieldClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision flux(\textcolor{red}{\textless double precision\textgreater} wavelength\argin,\textcol type((treeNode))\textgreater} node\arginout)` Return the flux (in units of ergs cm^2 s^{-1} Hz^{-1} ster^{-1}) of the given radiation field.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`double precision integrateOverCrossSection(\textcolor{red}{\textless double precision\textgreater} wavelengthRange\argin,\textcolor{red}{\textless double precision\textgreater} crossSectionFunction, type((treeNode))\textgreater} node\arginout)` Integrates the flux (in units of ergs cm^2 s^{-1} Hz^{-1} ster^{-1}) of the given radiation structure between the wavelengths given in `wavelengthRange` over a cross section specified by the function `crossSectionFunction`.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

radiationFieldCosmicMicrowaveBackground

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((radiationFieldClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision flux(\textcolor{red}{\textless double precision\textgreater} wavelength\argin,\textcol type((treeNode))\textgreater} node\arginout)` Return the flux (in units of ergs cm^2 s^{-1} Hz^{-1} ster^{-1}) of the given radiation field.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`double precision integrateOverCrossSection(\textcolor{red}{\textless double precision\textgreater} wavelengthRange\argin,\textcolor{red}{\textless double precision\textgreater} crossSectionFunction, type((treeNode))\textgreater} node\arginout)` Integrates the flux (in units of ergs cm^2 s^{-1} Hz^{-1} ster^{-1}) of the given radiation structure between the wavelengths given in `wavelengthRange` over a cross section specified by the function `crossSectionFunction`.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`<double> temperature()` Return the temperature of te black-body radiation field.

`<void> timeSet(<double> time→)` Set the time for the radiation field.

radiationFieldIntergalacticBackground

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((radiationFieldClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision flux(\textcolor{red}{\textless double precision\textgreater} wavelength\argin,\textcol type((treeNode))\textgreater} node\arginout)` Return the flux (in units of ergs cm^2 s^{-1} Hz^{-1} ster^{-1}) of the given radiation field.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`double precision integrateOverCrossSection(\textcolor{red}{\textless double precision\textgreater} wavelengthRange\argin,\textcolor{red}{\textless double precision\textgreater} crossSectionFunction, type((treeNode))\textgreater} node\arginout)` Integrates the flux (in units of ergs cm^2 s^{-1} Hz^{-1} ster^{-1}) of the given radiation structure between the wavelengths given in `wavelengthRange` over a cross section specified by the function `crossSectionFunction`.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`<void> timeSet(<double> time→)` Set the time for the radiation field.

radiationFieldIntergalacticBackgroundFile

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((radiationFieldClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision flux(\textcolor{red}{\textless double precision\textgreater} wavelength\argin,\textcol type((treeNode))\textgreater} node\arginout)` Return the flux (in units of ergs cm^2 s^{-1} Hz^{-1} ster^{-1}) of the given radiation field.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges`
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`double precision integrateOverCrossSection(\textcolor{red}{\textless double precision\textgreater} wavelengthRange\argin,\textcolor{red}{\textless double precision\textgreater} crossSectionFunction, type((treeNode))\textgreater} node\arginout)` Integrates the flux (in units of ergs cm^2 s^{-1} Hz^{-1} ster^{-1}) of the given radiation structure between the wavelengths given in `wavelengthRange` over a cross section specified by the function `crossSectionFunction`.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`<void> timeSet(<double> time→)` Set the time for the radiation field.

radiationFieldIntergalacticBackgroundInternal

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((radiationFieldClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision flux(\textcolor{red}{\textless double precision\textgreater} wavelength\argin,\textcol type((treeNode))\textgreater} node\arginout)` Return the flux (in units of ergs cm^2 s^{-1} Hz^{-1} ster^{-1}) of the given radiation field.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges`
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`double precision integrateOverCrossSection(\textcolor{red}{\textless double precision\textgreater} wavelengthRange\argin,\textcolor{red}{\textless double precision\textgreater} crossSectionFunction, type((treeNode))\textgreater} node\arginout)` Integrates the flux (in units of ergs cm^2 s^{-1} Hz^{-1} ster^{-1}) of the given radiation structure between the wavelengths given in `wavelengthRange` over a cross section specified by the function `crossSectionFunction`.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

<void> timeSet(<double> time→) Set the time for the radiation field.

radiationFieldNull

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((radiationFieldClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision flux(\textcolor{red}{\textless double precision\textgreater} wavelength\argin,\textcol type((treeNode))\textgreater} node\arginout) Return the flux (in units of ergs cm^2 s^{-1} Hz^{-1} ster^{-1}) of the given radiation field.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

double precision integrateOverCrossSection(\textcolor{red}{\textless double precision\textgreater} wavelengthRange\argin,\textcolor{red}{\textless double precision\textgreater} crossSectionFunction, type((treeNode))\textgreater} node\arginout) Integrates the flux (in units of ergs cm^2 s^{-1} Hz^{-1} ster^{-1}) of the given radiation structure between the wavelengths given in wavelengthRange over a cross section specified by the function crossSectionFunction.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

radiationFieldSummation

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((radiationFieldClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision flux(\textcolor{red}{\textless double precision\textgreater} wavelength\argin,\textcol type((treeNode))\textgreater} node\arginout) Return the flux (in units of ergs cm^2 s^{-1} Hz^{-1} ster^{-1}) of the given radiation field.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

double precision integrateOverCrossSection(\textcolor{red}{\textless double precision\textgreater} wavelengthRange\argin,\textcolor{red}{\textless double precision\textgreater} crossSectionFunction, type((treeNode))\textgreater} node\arginout) Integrates the flux (in units of ergs cm^2 s^{-1} Hz^{-1} ster^{-1}) of the given radiation structure between the wavelengths given in wavelengthRange over a cross section specified by the function crossSectionFunction.

<logical> isDefault() Return true if this is the default object of this class.

<void> list(<*type(radiationFieldList)>) Return a list of all sub-components.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

ramPressureStrippingDisksClass

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((ramPressureStrippingDisksClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`double precision rateMassLoss(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Returns the rate of mass loss (in $M_\odot$ Gyr^{-1}) due to ram pressure stripping of the disk component of `node`.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

ramPressureStrippingDisksNull

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((ramPressureStrippingDisksClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`double precision rateMassLoss(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Returns the rate of mass loss (in $M_\odot$ Gyr^{-1}) due to ram pressure stripping of the disk component of `node`.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

ramPressureStrippingDisksSimple

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((ramPressureStrippingDisksClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`double precision rateMassLoss(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Returns the rate of mass loss (in $M_\odot$ Gyr^{-1}) due to ram pressure stripping of the disk component of `node`.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

ramPressureStrippingSpheroidsClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((ramPressureStrippingSpheroidsClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision rateMassLoss(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the rate of mass loss (in $M_\odot$ Gyr^{-1}) due to ram pressure stripping of the spheroid component of node.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

ramPressureStrippingSpheroidsNull

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((ramPressureStrippingSpheroidsClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision rateMassLoss(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the rate of mass loss (in $M_\odot$ Gyr^{-1}) due to ram pressure stripping of the spheroid component of node.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

ramPressureStrippingSpheroidsSimple

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((ramPressureStrippingSpheroidsClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision rateMassLoss(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)
Returns the rate of mass loss (in $M_\odot$ Gyr^{-1}) due to ram pressure stripping of the spheroid component of node.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

regEx

<void> destroy() Destroy the regex.

<logical> matches(<character(len=*)> string→) Return true if a regular expression matches the supplied string.

rootFinder

<void> destroy() Destroy the rootFinder object.

<double> find(<double> [rootGuess]|<double(2)> [rootRange]) Find the root of the function given an initial guess or range.

<logical> isInitialized() Return the initialization state of a rootFinder object.

<void> rangeExpand(<double> [rangeExpandUpward]→,<double> [rangeExpandDownward]→, <rangeExpand> [rangeExpandType]→, <double> [rangeUpwardLimit]→, <double> [rangeDownwardLimit]→, <rangeExpandSignExpect> [rangeExpandDownwardSignExpect]→, <rangeExpandSignExpect> [rangeExpandUpwa
Specify how the initial range will be expanded in a rootFinder object to bracket the root.

<void> rootFunction(<function(<double> x→)> rootFunction) Set the function that evaluates $f(x)$ to use in a rootFinder object.

<void> rootFunctionDerivative(<function(<double> x→)> rootFunction, <function(<double> x→)> rootFunctionDerivative, <void(<double> x→, <double> f→, <double> fdf→)> rootFunctionBoth)
Set the functions that evaluate $f(x)$ and derivatives to use in a rootFinder object.

<void> tolerance(<double> [toleranceAbsolute]→, <double> [toleranceRelative]→) Set the tolerance to use in a rootFinder object.

<void> type(<type(fgsl_root_fsolver_type)> solverType→) Set the type of algorithm to use in a rootFinder object.

<void> typeDerivative(<type(fgsl_root_fdfsolver_type)> solverDerivativeType→) Set the type of algorithm to use in a rootFinder object in cases where the derivative of the function is available.

satelliteDynamicalFrictionChandrasekhar1943

double precision, dimension(3) acceleration(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the satellite acceleration due to dynamical friction for node (in units of km/s/Gyr).

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((satelliteDynamicalFrictionClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

satelliteDynamicalFrictionClass

double precision, dimension(3) acceleration(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the satellite acceleration due to dynamical friction for node (in units of km/s/Gyr).

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((satelliteDynamicalFrictionClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

satelliteDynamicalFrictionZero

`double precision, dimension(3) acceleration(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Returns the satellite acceleration due to dynamical friction for `node` (in units of km/s/Gyr).

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((satelliteDynamicalFrictionClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

satelliteMergingTimescalesBoylanKolchin2008

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((satelliteMergingTimescalesClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision timeUntilMerging(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless type((keplerOrbit))\textgreater} orbit\arginout) Return the time (in Gyr) until the satellite will merge with its host given the current orbit.

satelliteMergingTimescalesClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((satelliteMergingTimescalesClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision timeUntilMerging(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless type((keplerOrbit))\textgreater} orbit\arginout) Return the time (in Gyr) until the satellite will merge with its host given the current orbit.

satelliteMergingTimescalesInfinite

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((satelliteMergingTimescalesClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision timeUntilMerging(\textcolor{red}{\textless type((treeNode))\textgreater}
node\arginout,\textcolor{red}{\textless type((keplerOrbit))\textgreater} orbit\arginout)
Return the time (in Gyr) until the satellite will merge with its host given the current orbit.

satelliteMergingTimescalesJiang2008

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((satelliteMergingTimescalesClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision timeUntilMerging(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless type((keplerOrbit))\textgreater} orbit\arginout) Return the time (in Gyr) until the satellite will merge with its host given the current orbit.

satelliteMergingTimescalesLaceyCole1993

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((satelliteMergingTimescalesClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision timeUntilMerging(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless type((keplerOrbit))\textgreater} orbit\arginout) Return the time (in Gyr) until the satellite will merge with its host given the current orbit.

double precision timeUntilMergingMassDependence() Return the mass-dependent part of the time (in Gyr) until the satellite will merge with its host.

satelliteMergingTimescalesLaceyCole1993Tormen

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((satelliteMergingTimescalesClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision timeUntilMerging(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless type((keplerOrbit))\textgreater} orbit\arginout) Return the time (in Gyr) until the satellite will merge with its host given the current orbit.

double precision timeUntilMergingMassDependence() Return the mass-dependent part of the time (in Gyr) until the satellite will merge with its host.

satelliteMergingTimescalesPreset

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((satelliteMergingTimescalesClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision timeUntilMerging(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless type((keplerOrbit))\textgreater} orbit\arginout) Return the time (in Gyr) until the satellite will merge with its host given the current orbit.

satelliteMergingTimescalesRandom

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((satelliteMergingTimescalesClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision timeUntilMerging(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless type((keplerOrbit))\textgreater} orbit\arginout) Return the time (in Gyr) until the satellite will merge with its host given the current orbit.

satelliteMergingTimescalesVillalobos2013

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((satelliteMergingTimescalesClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision timeUntilMerging(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless type((keplerOrbit))\textgreater} orbit\arginout) Return the time (in Gyr) until the satellite will merge with its host given the current orbit.

satelliteMergingTimescalesWetzelWhite2010

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((satelliteMergingTimescalesClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision timeUntilMerging(\textcolor{red}{\textless type((treeNode))\textgreater}
node\arginout,\textcolor{red}{\textless type((keplerOrbit))\textgreater} orbit\arginout)
Return the time (in Gyr) until the satellite will merge with its host given the current orbit.

satelliteMergingTimescalesZero

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((satelliteMergingTimescalesClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision timeUntilMerging(\textcolor{red}{\textless type((treeNode))\textgreater}
node\arginout,\textcolor{red}{\textless type((keplerOrbit))\textgreater} orbit\arginout)
Return the time (in Gyr) until the satellite will merge with its host given the current orbit.

satelliteOrphanDistributionClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((satelliteOrphanDistributionClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision extent(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)
The maximum extent of the distribution, i.e. the radius of a sphere centered on the host halo which encompasses all orphan satellites.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision, dimension(3) position(\textcolor{red}{\textless type((treeNode))\textgreater}
node\arginout) Return the position of the given orphan in physical coordinates.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision, dimension(3) velocity(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Return the peculiar velocity of the given orphan in physical coordinates.

satelliteOrphanDistributionRandomIsotropic

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((satelliteOrphanDistributionClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision extent(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) The maximum extent of the distribution, i.e. the radius of a sphere centered on the host halo which encompasses all orphan satellites.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<double> inverseCumulativeMassFunctionRadial(textcolorred<type(treeNode)> node↔, <double> fraction→) Return the radius enclosing the given fraction of the orphan satellite population.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision, dimension(3) position(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Return the position of the given orphan in physical coordinates.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision, dimension(3) velocity(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Return the peculiar velocity of the given orphan in physical coordinates.

`<double> velocityDispersion(textcolorred<type(treeNode)> node↔)` Return the 1-D velocity dispersion of the orphan satellite population.

satelliteOrphanDistributionTraceDarkMatter

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((satelliteOrphanDistributionClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision extent(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` The maximum extent of the distribution, i.e. the radius of a sphere centered on the host halo which encompasses all orphan satellites.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<double> inverseCumulativeMassFunctionRadial(textcolorred<type(treeNode)> node↔, <double> fraction→)` Return the radius enclosing the given fraction of the orphan satellite population.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`double precision, dimension(3) position(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Return the position of the given orphan in physical coordinates.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision, dimension(3) velocity(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Return the peculiar velocity of the given orphan in physical coordinates.

<double> velocityDispersion(textcolorred<type(treeNode)> node↔) Return the 1-D velocity dispersion of the orphan satellite population.

satelliteTidalFieldClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((satelliteTidalFieldClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision tidalTensorRadial(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the radial component, $\Phi_{\rm rr}$, of the tidal tensor, $\Phi_{\rm ab}$.

satelliteTidalFieldNull

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((satelliteTidalFieldClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision tidalTensorRadial(\textcolor{red}{\textless type((treeNode))\textgreater}
node\arginout) Returns the radial component, $\Phi_{\rm rr}$, of the tidal tensor, $\Phi_{\rm ab}$.

satelliteTidalFieldSphericalSymmetry

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((satelliteTidalFieldClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision tidalTensorRadial(\textcolor{red}{\textless type((treeNode))\textgreater}
node\arginout) Returns the radial component, $\Phi_{\rm rr}$, of the tidal tensor, $\Phi_{\rm ab}$.

satelliteTidalHeatingRateClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((satelliteTidalHeatingRateClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

double precision heatingRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)
Return the satellite tidal heating rate for node (in units of $(\mathrm{km/s/Mpc})^2/\mathrm{Gyr}$).

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

satelliteTidalHeatingRateGnedin1999

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((satelliteTidalHeatingRateClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

double precision heatingRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Return the satellite tidal heating rate for node (in units of $(\mathrm{km/s/Mpc})^2/\mathrm{Gyr}$).

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

satelliteTidalHeatingRateZero

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((satelliteTidalHeatingRateClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`double precision heatingRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Return the satellite tidal heating rate for `node` (in units of $(\mathrm{km/s/Mpc})^2/\mathrm{Gyr}$).

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

satelliteTidalStrippingClass

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((satelliteTidalStrippingClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision massLossRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Returns the rate of tidal mass loss for `node` (in units of $M_\odot/\mathrm{Gyr}$).

`type(varying_string) objectType()` Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

satelliteTidalStrippingZentner2005

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

<void> calculationReset(<type(table)> node↔) Reset memoized calculations.

void deepCopy(\textcolor{red}{\textless class((satelliteTidalStrippingClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision massLossRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the rate of tidal mass loss for node (in units of $M_\odot$/Gyr).

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

satelliteTidalStrippingZero

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((satelliteTidalStrippingClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision massLossRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Returns the rate of tidal mass loss for `node` (in units of $M_\odot$/Gyr).

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

semaphore

`<void> close()` Close the semaphore.

`<void> post()` Post to (i.e. release) a semaphore.

`<void> unlink()` Unlink a semaphore.

`<void> wait()` Wait for a semaphore to become available.

starFormationExpulsiveFeedbackDisksClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((starFormationExpulsiveFeedbackDisksClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision outflowRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\tex double precision\textgreater} rateEnergyInput\argin,\textcolor{red}{\textless double precision\textgreater} rateStarFormation\argin) Returns the outflow rate due to star formation in the disk component of node in units of $M_\odot$/Gyr.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

starFormationExpulsiveFeedbackDisksSuperWind

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((starFormationExpulsiveFeedbackDisksClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision outflowRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\tex
double precision\textgreater} rateEnergyInput\argin,\textcolor{red}{\textless double
precision\textgreater} rateStarFormation\argin) Returns the outflow rate due to star formation in the disk component of node in units of $M_\odot$/Gyr.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

starFormationExpulsiveFeedbackDisksZero

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((starFormationExpulsiveFeedbackDisksClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision outflowRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\tex
double precision\textgreater} rateEnergyInput\argin,\textcolor{red}{\textless double
precision\textgreater} rateStarFormation\argin) Returns the outflow rate due to star formation in the disk component of node in units of $M_\odot$/Gyr.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

starFormationExpulsiveFeedbackSpheroidsClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((starFormationExpulsiveFeedbackSpheroidsClass))\textgreat destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision outflowRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\tex double precision\textgreater} rateEnergyInput\argin,\textcolor{red}{\textless double precision\textgreater} rateStarFormation\argin) Returns the outflow rate due to star formation in the spheroid component of node in units of $M_\odot$/Gyr.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

starFormationExpulsiveFeedbackSpheroidsSuperWind

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((starFormationExpulsiveFeedbackSpheroidsClass))\textgreat destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`double precision outflowRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\tex double precision\textgreater} rateEnergyInput\argin,\textcolor{red}{\textless double precision\textgreater} rateStarFormation\argin)` Returns the outflow rate due to star formation in the spheroid component of `node` in units of $M_\odot$/Gyr.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

starFormationExpulsiveFeedbackSpheroidsZero

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((starFormationExpulsiveFeedbackSpheroidsClass))\textgreat destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision outflowRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\tex
double precision\textgreater} rateEnergyInput\argin,\textcolor{red}{\textless double
precision\textgreater} rateStarFormation\argin) Returns the outflow rate due to star formation in the spheroid component of node in units of $M_\odot$/Gyr.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

starFormationFeedbackDisksClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((starFormationFeedbackDisksClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision outflowRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\tex
double precision\textgreater} rateEnergyInput\argin,\textcolor{red}{\textless double
precision\textgreater} rateStarFormation\argin) Returns the outflow rate due to star formation in the disk component of node in units of $M_\odot$/Gyr.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

starFormationFeedbackDisksCreasey2012

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((starFormationFeedbackDisksClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision outflowRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\tex double precision\textgreater} rateEnergyInput\argin,\textcolor{red}{\textless double precision\textgreater} rateStarFormation\argin) Returns the outflow rate due to star formation in the disk component of node in units of $M_\odot$/Gyr.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

starFormationFeedbackDisksFixed

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((starFormationFeedbackDisksClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision outflowRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\tex
double precision\textgreater} rateEnergyInput\argin,\textcolor{red}{\textless double
precision\textgreater} rateStarFormation\argin) Returns the outflow rate due to star formation in the disk component of node in units of $M_\odot$/Gyr.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

starFormationFeedbackDisksHaloScaling

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((starFormationFeedbackDisksClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision outflowRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\tex
double precision\textgreater} rateEnergyInput\argin,\textcolor{red}{\textless double
precision\textgreater} rateStarFormation\argin) Returns the outflow rate due to star formation in the disk component of node in units of $M_\odot$/Gyr.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

starFormationFeedbackDisksPowerLaw

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((starFormationFeedbackDisksClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision outflowRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\tex
double precision\textgreater} rateEnergyInput\argin,\textcolor{red}{\textless double
precision\textgreater} rateStarFormation\argin) Returns the outflow rate due to star formation in the disk component of node in units of $M_\odot$/Gyr.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

<double> velocityCharacteristic(<type(treeNode)> node↔) Return the characteristic velocity for power law disk feedback models.

starFormationFeedbackDisksPowerLawRedshiftScaling

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((starFormationFeedbackDisksClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\textlogical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision outflowRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textdouble precision\textgreater} rateEnergyInput\argin,\textcolor{red}{\textless double precision\textgreater} rateStarFormation\argin) Returns the outflow rate due to star formation in the disk component of node in units of $M_\odot$/Gyr.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

<double> velocityCharacteristic(<type(treeNode)> node↔) Return the characteristic velocity for power law disk feedback models.

starFormationFeedbackDisksVlctyMxSclng

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((starFormationFeedbackDisksClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision outflowRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\tex double precision\textgreater} rateEnergyInput\argin,\textcolor{red}{\textless double precision\textgreater} rateStarFormation\argin) Returns the outflow rate due to star formation in the disk component of node in units of $M_\odot$/Gyr.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

starFormationFeedbackSpheroidsClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((starFormationFeedbackSpheroidsClass))\textgreater} destination\arginout) Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`double precision outflowRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\tex double precision\textgreater} rateEnergyInput\argin,\textcolor{red}{\textless double precision\textgreater} rateStarFormation\argin)` Returns the outflow rate due to star formation in the spheroid component of `node` in units of $M_\odot$/Gyr.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

starFormationFeedbackSpheroidsFixed

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((starFormationFeedbackSpheroidsClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`double precision outflowRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\tex double precision\textgreater} rateEnergyInput\argin,\textcolor{red}{\textless double precision\textgreater} rateStarFormation\argin)` Returns the outflow rate due to star formation in the spheroid component of `node` in units of $M_\odot$/Gyr.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

starFormationFeedbackSpheroidsPowerLaw

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((starFormationFeedbackSpheroidsClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision outflowRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\tex double precision\textgreater} rateEnergyInput\argin,\textcolor{red}{\textless double precision\textgreater} rateStarFormation\argin) Returns the outflow rate due to star formation in the spheroid component of node in units of $M_\odot$/Gyr.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

<double> velocityCharacteristic(<type(treeNode)> node↔) Return the characteristic velocity for power law spheroid feedback models.

starFormationFeedbackSpheroidsPowerLawRedshiftScaling

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((starFormationFeedbackSpheroidsClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision outflowRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\tex double precision\textgreater} rateEnergyInput\argin,\textcolor{red}{\textless double precision\textgreater} rateStarFormation\argin) Returns the outflow rate due to star formation in the spheroid component of node in units of $M_\odot$/Gyr.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

<double> velocityCharacteristic(<type(treeNode)> node↔) Return the characteristic velocity for power law spheroid feedback models.

starFormationFeedbackSpheroidsVlctyMxSclng

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((starFormationFeedbackSpheroidsClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`double precision outflowRate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\tex double precision\textgreater} rateEnergyInput\argin,\textcolor{red}{\textless double precision\textgreater} rateStarFormation\argin)` Returns the outflow rate due to star formation in the spheroid component of `node` in units of $M_\odot$/Gyr.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

starFormationHistoryClass

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void create(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\text type((history ))\textgreater} historyStarFormation\arginout,\textcolor{red}{\textless double precision\textgreater} timeBegin\argin)` Create the star formation history object.

`void deepCopy(\textcolor{red}{\textless class((starFormationHistoryClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

void output(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\tex
logical\textgreater} nodePassesFilter\argin,\textcolor{red}{\textless type((history
))\textgreater} historyStarFormation\arginout,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} indexOutput\argin,\textcolor{red}{\textless integer((kind=kind_-
int8))\textgreater} indexTree\argin,\textcolor{red}{\textless character((len=*))\textgreater}
labelComponent\argin) Output the star formation history.

void rate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textl
type((history))\textgreater} historyStarFormation\arginout,\textcolor{red}{\textless
type((abundances))\textgreater} abundancesFuel\argin,\textcolor{red}{\textless double
precision\textgreater} rateStarFormation\argin) Record the rate of star formation in this
history.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the
new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void scales(\textcolor{red}{\textless type((history))\textgreater} historyStarFormation\arginout,\text
double precision\textgreater} massStellar\argin,\textcolor{red}{\textless type((abundances))\textgr
abundancesStellar\argin) Set ODE solver absolute scales for a star formation history object.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

starFormationHistoryInSitu

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names al-
lowed for this object.

void autoHook() Insert any event hooks required by this object.

void create(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\text
type((history))\textgreater} historyStarFormation\arginout,\textcolor{red}{\textless
double precision\textgreater} timeBegin\argin) Create the star formation history object.

void deepCopy(\textcolor{red}{\textless class((starFormationHistoryClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which
could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

<void> make(<type(history)> historyStarFormation↔, <double> timeBegin→, <double> timeEnd→,
<double(:)> [timesCurrent]) Make the star formation history.

type(varying_string) objectType() Return the type of the object.

void output(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\tex
logical\textgreater} nodePassesFilter\argin,\textcolor{red}{\textless type((history
))\textgreater} historyStarFormation\arginout,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} indexOutput\argin,\textcolor{red}{\textless integer((kind=kind_-
int8))\textgreater} indexTree\argin,\textcolor{red}{\textless character((len=*))\textgreater}
labelComponent\argin) Output the star formation history.

void rate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textl
type((history))\textgreater} historyStarFormation\arginout,\textcolor{red}{\textless
type((abundances))\textgreater} abundancesFuel\argin,\textcolor{red}{\textless double
precision\textgreater} rateStarFormation\argin) Record the rate of star formation in this
history.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the
new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void scales(\textcolor{red}{\textless type((history))\textgreater} historyStarFormation\arginout,\text
double precision\textgreater} massStellar\argin,\textcolor{red}{\textless type((abundances))\textgr
abundancesStellar\argin) Set ODE solver absolute scales for a star formation history object.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

starFormationHistoryMetallicitySplit

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names al-
lowed for this object.

void autoHook() Insert any event hooks required by this object.

void create(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\text
type((history))\textgreater} historyStarFormation\arginout,\textcolor{red}{\textless
double precision\textgreater} timeBegin\argin) Create the star formation history object.

void deepCopy(\textcolor{red}{\textless class((starFormationHistoryClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

<void> make(<type(history)> historyStarFormation↔, <double> timeBegin→, <double> timeEnd→, <double(:)> [timesCurrent]) Make the star formation history.

type(varying_string) objectType() Return the type of the object.

void output(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\tex logical\textgreater} nodePassesFilter\argin,\textcolor{red}{\textless type((history))\textgreater} historyStarFormation\arginout,\textcolor{red}{\textless integer((c_-size_t))\textgreater} indexOutput\argin,\textcolor{red}{\textless integer((kind=kind_-int8))\textgreater} indexTree\argin,\textcolor{red}{\textless character((len=*))\textgreater} labelComponent\argin) Output the star formation history.

void rate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textl type((history))\textgreater} historyStarFormation\arginout,\textcolor{red}{\textless type((abundances))\textgreater} abundancesFuel\argin,\textcolor{red}{\textless double precision\textgreater} rateStarFormation\argin) Record the rate of star formation in this history.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void scales(\textcolor{red}{\textless type((history))\textgreater} historyStarFormation\arginout,\text double precision\textgreater} massStellar\argin,\textcolor{red}{\textless type((abundances))\textgr abundancesStellar\argin) Set ODE solver absolute scales for a star formation history object.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

starFormationHistoryNull

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void create(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\text
type((history))\textgreater} historyStarFormation\arginout,\textcolor{red}{\textless
double precision\textgreater} timeBegin\argin) Create the star formation history object.

void deepCopy(\textcolor{red}{\textless class((starFormationHistoryClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which
could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

void output(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\tex
logical\textgreater} nodePassesFilter\argin,\textcolor{red}{\textless type((history
))\textgreater} historyStarFormation\arginout,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} indexOutput\argin,\textcolor{red}{\textless integer((kind=kind_-
int8))\textgreater} indexTree\argin,\textcolor{red}{\textless character((len=*))\textgreater}
labelComponent\argin) Output the star formation history.

void rate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textl
type((history))\textgreater} historyStarFormation\arginout,\textcolor{red}{\textless
type((abundances))\textgreater} abundancesFuel\argin,\textcolor{red}{\textless double
precision\textgreater} rateStarFormation\argin) Record the rate of star formation in this
history.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the
new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void scales(\textcolor{red}{\textless type((history))\textgreater} historyStarFormation\arginout,\text
double precision\textgreater} massStellar\argin,\textcolor{red}{\textless type((abundances))\textgr
abundancesStellar\argin) Set ODE solver absolute scales for a star formation history object.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

starFormationRateSurfaceDensityDisksBlitz2006

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

<void> calculationReset(<type(table)> node↔) Reset memoized calculations.

void deepCopy(\textcolor{red}{\textless class((starFormationRateSurfaceDensityDisksClass))\textgreater]
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

double precision, allocatable, dimension(:,:) intervals(\textcolor{red}{\textless type((treeNode))\text
node\arginout,\textcolor{red}{\textless double precision\textgreater} radiusInner\argin,\textcolor{
double precision\textgreater} radiusOuter\argin) Return a set of integration intervals to use when integrating over the surface density of star formation rate.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision rate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{
double precision\textgreater} radius\argin) Returns the star formation rate surface density (in $M_\odot$ Gyr^{-1} Mpc^{-2}) in the disk component of thisNode at the given radius.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

logical unchanged(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)
Return true if the surface density rate of star formation is unchanged since the previous evaluation.

starFormationRateSurfaceDensityDisksClass

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((starFormationRateSurfaceDensityDisksClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`double precision, allocatable, dimension(:,:) intervals(\textcolor{red}{\textless type((treeNode))\text node\arginout,\textcolor{red}{\textless double precision\textgreater} radiusInner\argin,\textcolor{ double precision\textgreater} radiusOuter\argin)` Return a set of integration intervals to use when integrating over the surface density of star formation rate.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`double precision rate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{ double precision\textgreater} radius\argin)` Returns the star formation rate surface density (in $M_\odot$ Gyr^{-1} Mpc^{-2}) in the disk component of `thisNode` at the given `radius`.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`logical unchanged(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Return true if the surface density rate of star formation is unchanged since the previous evaluation.

starFormationRateSurfaceDensityDisksExtendedSchmidt

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

<void> calculationReset(<type(table)> node↔) Reset memoized calculations.

void deepCopy(\textcolor{red}{\textless class((starFormationRateSurfaceDensityDisksClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

double precision, allocatable, dimension(:,:) intervals(\textcolor{red}{\textless type((treeNode))\text node\arginout,\textcolor{red}{\textless double precision\textgreater} radiusInner\argin,\textcolor{ double precision\textgreater} radiusOuter\argin) Return a set of integration intervals to use when integrating over the surface density of star formation rate.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision rate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{ double precision\textgreater} radius\argin) Returns the star formation rate surface density (in $M_\odot$ Gyr^{-1} Mpc^{-2}) in the disk component of thisNode at the given radius.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

logical unchanged(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Return true if the surface density rate of star formation is unchanged since the previous evaluation.

starFormationRateSurfaceDensityDisksKennicuttSchmidt

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

<void> calculationReset(<type(table)> node↔) Reset memoized calculations.

void deepCopy(\textcolor{red}{\textless class((starFormationRateSurfaceDensityDisksClass))\textgreater} destination\arginout) Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`double precision, allocatable, dimension(:,:) intervals(\textcolor{red}{\textless type((treeNode))\text node\arginout,\textcolor{red}{\textless double precision\textgreater} radiusInner\argin,\textcolor{ double precision\textgreater} radiusOuter\argin)` Return a set of integration intervals to use when integrating over the surface density of star formation rate.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`double precision rate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{ double precision\textgreater} radius\argin)` Returns the star formation rate surface density (in $M_\odot$ Gyr^{-1} Mpc^{-2}) in the disk component of `thisNode` at the given `radius`.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`logical unchanged(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Return true if the surface density rate of star formation is unchanged since the previous evaluation.

starFormationRateSurfaceDensityDisksKrumholz2009

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`<void> calculationReset(<type(table)> node↔)` Reset memoized calculations.

`<void> computeFactors()` Compute constant factors required.

`void deepCopy(\textcolor{red}{\textless class((starFormationRateSurfaceDensityDisksClass))\textgreater destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

double precision, allocatable, dimension(:,:) intervals(\textcolor{red}{\textless type((treeNode))\text
node\arginout,\textcolor{red}{\textless double precision\textgreater} radiusInner\argin,\textcolor{
double precision\textgreater} radiusOuter\argin) Return a set of integration intervals to use when integrating over the surface density of star formation rate.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision rate(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{
double precision\textgreater} radius\argin) Returns the star formation rate surface density (in $M_\odot$ Gyr^{-1} Mpc^{-2}) in the disk component of thisNode at the given radius.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

<void> surfaceDensityFactors() Compute surface density factors required.

logical unchanged(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)
Return true if the surface density rate of star formation is unchanged since the previous evaluation.

starFormationTimescaleDisksBaugh2005

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((starFormationTimescaleDisksClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision timescale(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the timescale (in Gyr) for star formation in the disk component of node.

starFormationTimescaleDisksClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((starFormationTimescaleDisksClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision timescale(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the timescale (in Gyr) for star formation in the disk component of node.

starFormationTimescaleDisksDynamicalTime

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((starFormationTimescaleDisksClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision timescale(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)
Returns the timescale (in Gyr) for star formation in the disk component of node.

starFormationTimescaleDisksFixed

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((starFormationTimescaleDisksClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision timescale(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)
Returns the timescale (in Gyr) for star formation in the disk component of node.

starFormationTimescaleDisksHaloScaling

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

<void> calculationReset(<type(table)> node↔) Reset memoized calculations.

void deepCopy(\textcolor{red}{\textless class((starFormationTimescaleDisksClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision timescale(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the timescale (in Gyr) for star formation in the disk component of node.

starFormationTimescaleDisksIntgrtdSurfaceDensity

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((starFormationTimescaleDisksClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision timescale(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the timescale (in Gyr) for star formation in the disk component of node.

starFormationTimescaleDisksVelocityMaxScaling

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

<void> calculationReset(<type(table)> node↔) Reset memoized calculations.

void deepCopy(\textcolor{red}{\textless class((starFormationTimescaleDisksClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision timescale(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)
Returns the timescale (in Gyr) for star formation in the disk component of node.

starFormationTimescaleSpheroidsClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((starFormationTimescaleSpheroidsClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision timescale(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)
Returns the timescale (in Gyr) for star formation in the spheroid component of node.

starFormationTimescaleSpheroidsDynamicalTime

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((starFormationTimescaleSpheroidsClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision timescale(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Returns the timescale (in Gyr) for star formation in the spheroid component of `node`.

starFormationTimescaleSpheroidsVelocityMaxScaling

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`<void> calculationReset(<type(table)> node↔)` Reset memoized calculations.

`void deepCopy(\textcolor{red}{\textless class((starFormationTimescaleSpheroidsClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision timescale(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Returns the timescale (in Gyr) for star formation in the spheroid component of `node`.

stellarAstrophysicsClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((stellarAstrophysicsClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision lifetime(\textcolor{red}{\textless double precision\textgreater} massInitial\argin,\te
double precision\textgreater} metallicity\argin) Returns the lifetime of a star of given massInitial and metallicity.

double precision massEjected(\textcolor{red}{\textless double precision\textgreater} massInitial\argin,
double precision\textgreater} metallicity\argin) Returns the mass ejected by a star of given massInitial and metallicity.

double precision massInitial(\textcolor{red}{\textless double precision\textgreater} lifetime\argin,\te
double precision\textgreater} metallicity\argin) Returns the initial mass of a star of given lifetime and metallicity.

double precision massYield(\textcolor{red}{\textless double precision\textgreater} massInitial\argin,\t
double precision\textgreater} metallicity\argin,\textcolor{red}{\textless integer\textgreater}
atomIndex\argin) Returns the metal mass yielded by a star of given massInitial and metallicity.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

stellarAstrophysicsFile

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((stellarAstrophysicsClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision lifetime(\textcolor{red}{\textless double precision\textgreater} massInitial\argin,\te double precision\textgreater} metallicity\argin)` Returns the lifetime of a star of given `massInitial` and `metallicity`.

`double precision massEjected(\textcolor{red}{\textless double precision\textgreater} massInitial\argin, double precision\textgreater} metallicity\argin)` Returns the mass ejected by a star of given `massInitial` and `metallicity`.

`double precision massInitial(\textcolor{red}{\textless double precision\textgreater} lifetime\argin,\te double precision\textgreater} metallicity\argin)` Returns the initial mass of a star of given `lifetime` and `metallicity`.

`double precision massYield(\textcolor{red}{\textless double precision\textgreater} massInitial\argin,\t double precision\textgreater} metallicity\argin,\textcolor{red}{\textless integer\textgreater} atomIndex\argin)` Returns the metal mass yielded by a star of given `massInitial` and `metallicity`.

`type(varying_string) objectType()` Return the type of the object.

`<void> read()` Read stellar astrophysics data from file.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

stellarFeedbackClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((stellarFeedbackClass))\textgreater} destination\arginout
Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision energyInputCumulative(\textcolor{red}{\textless double precision\textgreater}
initialMass\argin,\textcolor{red}{\textless double precision\textgreater} age\argin,\textcolor{red}
double precision\textgreater} metallicity\argin) Return the cumulative energy input from a stellar population of the given initialMass, age, and metallicity.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

stellarFeedbackStandard

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((stellarFeedbackClass))\textgreater} destination\arginout
Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision energyInputCumulative(\textcolor{red}{\textless double precision\textgreater} initialMass\argin,\textcolor{red}{\textless double precision\textgreater} age\argin,\textcolor{red}{\textless double precision\textgreater} metallicity\argin) Return the cumulative energy input from a stellar population of the given initialMass, age, and metallicity.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

stellarLuminosities

<type(stellarLuminosities)> add(<type(stellarLuminosities)> stellarLuminosities2→) Add two stellarLuminosities.

<void> builder(<*type(node)> stellarLuminositiesDefinition→) Build a stellar luminosities object from a provided XML description.

<void> deserialize(<double(:)> array→) Deserialize a stellar luminosities object from an array.

<void> destroy() Destroy a stellar luminosities object.

<type(stellarLuminosities)> divide(<double> divisor→) Divide stellar luminosities by a scalar.

<void> dump() Dump a stellar luminosities object.

<void> dumpRaw(<integer> fileHandle→) Dump a stellar luminosities object to binary.

<void> increment(<type(stellarLuminosities)> addStellarLuminosities→) Increment a stellar luminosities object.

<integer> index(<type(varying_string)> name→) Return the index to a luminosity specified by name.

<logical> isOutput(<integer> index→, <double> time→) Return true if the indexed luminosity is to be output at the given time.

<logical> isZero() Return true if a stellar luminosities object is zero.

<double> luminosity(<integer> index→) Return the i^{th} luminosity.

<integer> luminosityCount(<logical> [unmapped]) Return the total number of luminosities tracked. If unmapped is true, then the number of luminosities prior to mapping is returned.

<integer> luminosityOutputCount(<double> time→) Return the number of luminosities to be output at the given time.

<type(stellarLuminosities)> multiply(<double> multiplier→) Multiply stellar luminosities by a scalar.

<type(varying_string)> name(<integer> index→) Return the name of a luminosity specified by index.

<integer(c_size_t) > nonStaticSizeOf() Returns the size of any non-static components of the type.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(:,:)> integerBuffer↔, <integer>doubleProperty↔, <integer> doubleBufferCount↔, <double(:,:)> doubleBuffer↔, <double> time→, <integer> instance→) Store a stellar luminosities object in the output buffers.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double> time→, <integer> instance→) Specify the count of a stellar luminosities object for output.

<void> outputNames(<integer> integerProperty↔, <char[*](:)> integerPropertyNames↔, <char[*](:)> integerPropertyComments↔, <double(:)> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <char[*](:)> doublePropertyNames↔, <char[*](:)> doublePropertyComments↔, <double(:)> doublePropertyUnitsSI↔, <double> time→, <integer> instance→) Specify the names of stellar luminosities object properties for output.

<void> postOutput(<double> time→) Store a stellar luminosities object in the output buffers.

<void> readRaw(<integer> fileHandle→) Read a stellar luminosities object from binary.

<void> reset() Reset a stellar luminosities object.

<void> serialize(<double(:)> array←) Serialize a stellar luminosities object to an array.

<integer> serializeCount() Return a count of the number of properties in a serialized stellar luminosities object.

<void> setLuminosities(<double> mass→,<class(stellarPopulationClass) stellarPopulation_↔,<double>currentTime→,<type(abundances)> fuelAbundances→) Set the luminosities using a single stellar population.

<void> setToUnity() Set a stellar luminosities object to unity.

<type(stellarLuminosities)> subtract(<type(stellarLuminosities)> stellarLuminosities2→) Subtract one abundance from another.

<void> truncate(<type(stellarLuminosities)> templateLuminosities→) Truncate the number of stellar luminosities stored to match that in the given templateLuminosities.

stellarPopulationClass

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((stellarPopulationClass))\textgreater} destination\argino` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`double precision rateEnergy(\textcolor{red}{\textless type((abundances))\textgreater} abundances_- \argin,\textcolor{red}{\textless double precision\textgreater} ageMinimum\argin,\textcolor{red}{\te double precision\textgreater} ageMaximum\argin)` Return the rate of energy input from this population.

`double precision rateRecycling(\textcolor{red}{\textless type((abundances))\textgreater} abundances_ \argin,\textcolor{red}{\textless double precision\textgreater} ageMinimum\argin,\textco double precision\textgreater} ageMaximum\argin)` Return the rate of recycling from this population.

`double precision rateYield(\textcolor{red}{\textless type((abundances))\textgreater} abundances_- \argin,\textcolor{red}{\textless double precision\textgreater} ageMinimum\argin,\textcolor{red}{\te double precision\textgreater} ageMaximum\argin,\textcolor{red}{\textless integer\textgreater} elementIndex \argin)` Return the rate of element yield from this population.

`double precision recycledFractionInstantaneous()` Return the recycled fraction from this population in the instantaneous approximation.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`class(stellarPopulationSpectraClass) spectra()` Return at set of stellar spectra for this population.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

integer(c_size_t) uniqueID() Return the uniqueID corresponding to this population.

double precision yieldInstantaneous() Return the metal yield from this population in the instantaneous approximation.

stellarPopulationPropertiesClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((stellarPopulationPropertiesClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

integer historyCount() Return the number of stellar population property histories which must be stored.

void historyCreate(\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout,\textcolor{re type((history))\textgreater} history_\arginout) Create histories needed to store stellar population properties.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

void rates(\textcolor{red}{\textless double precision\textgreater} rateStarFormation \argin,\textcolor{ type((abundances))\textgreater} abundancesFuel \argin,\textcolor{red}{\textless class((nodeCompone))\textgreater} component \argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout,\textcolor{red}{\textless type((history))\textgreater} history_ \arginout,\textcolo double precision\textgreater} rateMassStellar\argout,\textcolor{red}{\textless double precision\textgreater} rateMassFuel\argout,\textcolor{red}{\textless double precision\textgreater} rateEnergyInput\argout,\textcolor{red}{\textless type((abundances))\textgreater} rateAbundancesFue type((abundances))\textgreater} rateAbundancesStellar \arginout,\textcolor{red}{\textless type((stellarLuminosities))\textgreater} rateLuminosityStellar \arginout,\textcolor{red}{\textless logical\textgreater} computeRateLuminosityStellar \argin) Returns rates of change of stellar population properties.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

`void scales(\textcolor{red}{\textless double precision\textgreater} massStellar \argin,\textcolor{red}{\textless type((abundances))\textgreater} abundancesStellar\argin,\textcolor{red}{\textless type((history ))\textgreater} history_ \arginout)` Return scaling factors of stellar population properties for an ODE solver.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

stellarPopulationPropertiesInstantaneous

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((stellarPopulationPropertiesClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`integer historyCount()` Return the number of stellar population property histories which must be stored.

`void historyCreate(\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout,\textcolor{re type((history ))\textgreater} history_\arginout)` Create histories needed to store stellar population properties.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`void rates(\textcolor{red}{\textless double precision\textgreater} rateStarFormation \argin,\textcolor type((abundances))\textgreater} abundancesFuel \argin,\textcolor{red}{\textless class((nodeCompone))\textgreater} component \argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout,\textcolor{red}{\textless type((history))\textgreater} history_ \arginout,\textcolo double precision\textgreater} rateMassStellar\argout,\textcolor{red}{\textless double precision\textgreater} rateMassFuel\argout,\textcolor{red}{\textless double precision\textgreater} rateEnergyInput\argout,\textcolor{red}{\textless type((abundances))\textgreater} rateAbundancesFue type((abundances))\textgreater} rateAbundancesStellar \arginout,\textcolor{red}{\textless type((stellarLuminosities))\textgreater} rateLuminosityStellar \arginout,\textcolor{red}{\textless logical\textgreater} computeRateLuminosityStellar \argin)` Returns rates of change of stellar population properties.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void scales(\textcolor{red}{\textless double precision\textgreater} massStellar \argin,\textcolor{red}{\textless type((abundances))\textgreater} abundancesStellar\argin,\textcolor{red}{\textless type((history))\textgreater} history_ \arginout) Return scaling factors of stellar population properties for an ODE solver.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

stellarPopulationPropertiesNoninstantaneous

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((stellarPopulationPropertiesClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\textlogical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

integer historyCount() Return the number of stellar population property histories which must be stored.

void historyCreate(\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout,\textcolor{retype((history))\textgreater} history_\arginout) Create histories needed to store stellar population properties.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

void rates(\textcolor{red}{\textless double precision\textgreater} rateStarFormation \argin,\textcolor{type((abundances))\textgreater} abundancesFuel \argin,\textcolor{red}{\textless class((nodeCompone))\textgreater} component \argin,\textcolor{red}{\textless type((treeNode))\textgreater} node \arginout,\textcolor{red}{\textless type((history))\textgreater} history_ \arginout,\textcolodouble precision\textgreater} rateMassStellar\argout,\textcolor{red}{\textless double precision\textgreater} rateMassFuel\argout,\textcolor{red}{\textless double precision\textgreater}

rateEnergyInput\argout,\textcolor{red}{\textless type((abundances))\textgreater} rateAbundancesFue
type((abundances))\textgreater} rateAbundancesStellar \arginout,\textcolor{red}{\textless
type((stellarLuminosities))\textgreater} rateLuminosityStellar \arginout,\textcolor{red}{\textless
logical\textgreater} computeRateLuminosityStellar \argin) Returns rates of change of stellar population properties.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void scales(\textcolor{red}{\textless double precision\textgreater} massStellar \argin,\textcolor{red}{
type((abundances))\textgreater} abundancesStellar\argin,\textcolor{red}{\textless type((history
))\textgreater} history_ \arginout) Return scaling factors of stellar population properties for an ODE solver.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

stellarPopulationSelectorClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((stellarPopulationSelectorClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

logical isStarFormationRateDependent() Return true if the selection of stellar population is dependent on star formation rate.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

class(stellarPopulationClass) select(\textcolor{red}{\textless double precision\textgreater} rateStarFormation\argin,\textcolor{red}{\textless type((abundances))\textgreater} abundances_ \argin,\textcolor{red}{\textless class((nodeComponent))\textgreater} component \argin) Return a stellar population.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

stellarPopulationSelectorDiskSpheroid

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((stellarPopulationSelectorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

logical isStarFormationRateDependent() Return true if the selection of stellar population is dependent on star formation rate.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

class(stellarPopulationClass) select(\textcolor{red}{\textless double precision\textgreater} rateStarFormation\argin,\textcolor{red}{\textless type((abundances))\textgreater} abundances_ \argin,\textcolor{red}{\textless class((nodeComponent))\textgreater} component \argin) Return a stellar population.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

stellarPopulationSelectorFixed

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((stellarPopulationSelectorClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

logical isStarFormationRateDependent() Return true if the selection of stellar population is dependent on star formation rate.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

class(stellarPopulationClass) select(\textcolor{red}{\textless double precision\textgreater} rateStarFormation\argin,\textcolor{red}{\textless type((abundances))\textgreater} abundances_ \argin,\textcolor{red}{\textless class((nodeComponent))\textgreater} component \argin) Return a stellar population.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

stellarPopulationSpectraClass

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((stellarPopulationSpectraClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision luminosity(\textcolor{red}{\textless type((abundances))\textgreater} abundancesStellar double precision\textgreater} age\argin,\textcolor{red}{\textless double precision\textgreater} wavelength\argin,\textcolor{red}{\textless integer\textgreater} status\argout)` Return the luminosity (in units of $L_\odot$ Hz^{-1}) for a stellar population, composition `abundances`, of the given `age` (in Gyr), at the specified `wavelength` (in Angstroms).

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`void tabulation(\textcolor{red}{\textless integer\textgreater} agesCount\argout,\textcolor{red}{\textle integer\textgreater} metallicitiesCount\argout,\textcolor{red}{\textless double precision\textgreat ages\argout,\textcolor{red}{\textless double precision\textgreater} metallicity\argout)` Return a tabulation of ages and metallicities at which stellar spectra should be tabulated.

`double precision wavelengthInterval(\textcolor{red}{\textless double precision\textgreater} wavelength\argin)` At a given wavelength, return the wavelength interval in the tabulation of wavelength for which stellar spectra are defined.

`void wavelengths(\textcolor{red}{\textless integer\textgreater} wavelengthsCount\argout,\textcolor{red} double precision\textgreater} wavelengths\argout)` Return a tabulation of wavelengths at which stellar spectra are defined.

stellarPopulationSpectraFile

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((stellarPopulationSpectraClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision luminosity(\textcolor{red}{\textless type((abundances))\textgreater} abundancesStellar double precision\textgreater} age\argin,\textcolor{red}{\textless double precision\textgreater} wavelength\argin,\textcolor{red}{\textless integer\textgreater} status\argout)` Return the luminosity (in units of $L_\odot$ Hz^{-1}) for a stellar population, composition `abundances`, of the given `age` (in Gyr), at the specified `wavelength` (in Angstroms).

`type(varying_string) objectType()` Return the type of the object.

`void readFile(<char(len=*)> fileName→)` Read the named stellar population spectra file.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`void tabulation(\textcolor{red}{\textless integer\textgreater} agesCount\argout,\textcolor{red}{\textle integer\textgreater} metallicitiesCount\argout,\textcolor{red}{\textless double precision\textgreat ages\argout,\textcolor{red}{\textless double precision\textgreater} metallicity\argout)` Return a tabulation of ages and metallicities at which stellar spectra should be tabulated.

`double precision wavelengthInterval(\textcolor{red}{\textless double precision\textgreater} wavelength\argin)` At a given wavelength, return the wavelength interval in the tabulation of wavelength for which stellar spectra are defined.

`void wavelengths(\textcolor{red}{\textless integer\textgreater} wavelengthsCount\argout,\textcolor{red} double precision\textgreater} wavelengths\argout)` Return a tabulation of wavelengths at which stellar spectra are defined.

stellarPopulationSpectraFSPS

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((stellarPopulationSpectraClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision luminosity(\textcolor{red}{\textless type((abundances))\textgreater} abundancesStellar double precision\textgreater} age\argin,\textcolor{red}{\textless double precision\textgreater} wavelength\argin,\textcolor{red}{\textless integer\textgreater} status\argout)` Return the luminosity (in units of $L_\odot$ Hz^{-1}) for a stellar population, composition `abundances`, of the given `age` (in Gyr), at the specified `wavelength` (in Angstroms).

`type(varying_string) objectType()` Return the type of the object.

`void readFile(<char(len=*)> fileName→)` Read the named stellar population spectra file.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`void tabulation(\textcolor{red}{\textless integer\textgreater} agesCount\argout,\textcolor{red}{\textle integer\textgreater} metallicitiesCount\argout,\textcolor{red}{\textless double precision\textgreat ages\argout,\textcolor{red}{\textless double precision\textgreater} metallicity\argout)` Return a tabulation of ages and metallicities at which stellar spectra should be tabulated.

`double precision wavelengthInterval(\textcolor{red}{\textless double precision\textgreater} wavelength\argin)` At a given wavelength, return the wavelength interval in the tabulation of wavelength for which stellar spectra are defined.

`void wavelengths(\textcolor{red}{\textless integer\textgreater} wavelengthsCount\argout,\textcolor{red} double precision\textgreater} wavelengths\argout)` Return a tabulation of wavelengths at which stellar spectra are defined.

stellarPopulationSpectraPostprocessorBuilderClass

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`class(stellarPopulationSpectraPostprocessorClass) build(\textcolor{red}{\textless type((varying_string))\textgreater} descriptor\argin)` Build and return a postprocessor.

`void deepCopy(\textcolor{red}{\textless class((stellarPopulationSpectraPostprocessorBuilderClass))\text destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_ size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_ size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

stellarPopulationSpectraPostprocessorBuilderLookup

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`class(stellarPopulationSpectraPostprocessorClass) build(\textcolor{red}{\textless type((varying_string))\textgreater} descriptor\argin)` Build and return a postprocessor.

`void deepCopy(\textcolor{red}{\textless class((stellarPopulationSpectraPostprocessorBuilderClass))\text destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

stellarPopulationSpectraPostprocessorClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((stellarPopulationSpectraPostprocessorClass))\textgreater
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision multiplier(\textcolor{red}{\textless double precision\textgreater} wavelength\argin,\t
double precision\textgreater} age\argin,\textcolor{red}{\textless double precision\textgreater}
redshift\argin) Return the multiplicative modification to the spectrum.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

stellarPopulationSpectraPostprocessorIdentity

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((stellarPopulationSpectraPostprocessorClass))\textgreater destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision multiplier(\textcolor{red}{\textless double precision\textgreater} wavelength\argin,\t double precision\textgreater} age\argin,\textcolor{red}{\textless double precision\textgreater} redshift\argin) Return the multiplicative modification to the spectrum.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

stellarPopulationSpectraPostprocessorInoue2014

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((stellarPopulationSpectraPostprocessorClass))\textgreater destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision multiplier(\textcolor{red}{\textless double precision\textgreater} wavelength\argin,\t double precision\textgreater} age\argin,\textcolor{red}{\textless double precision\textgreater} redshift\argin) Return the multiplicative modification to the spectrum.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

stellarPopulationSpectraPostprocessorLycSuppress

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((stellarPopulationSpectraPostprocessorClass))\textgreater destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision multiplier(\textcolor{red}{\textless double precision\textgreater} wavelength\argin,\t
double precision\textgreater} age\argin,\textcolor{red}{\textless double precision\textgreater}
redshift\argin) Return the multiplicative modification to the spectrum.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

stellarPopulationSpectraPostprocessorMadau1995

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((stellarPopulationSpectraPostprocessorClass))\textgreate
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision multiplier(\textcolor{red}{\textless double precision\textgreater} wavelength\argin,\t
double precision\textgreater} age\argin,\textcolor{red}{\textless double precision\textgreater}
redshift\argin) Return the multiplicative modification to the spectrum.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

stellarPopulationSpectraPostprocessorMeiksin2006

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((stellarPopulationSpectraPostprocessorClass))\textgreater destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision multiplier(\textcolor{red}{\textless double precision\textgreater} wavelength\argin,\t double precision\textgreater} age\argin,\textcolor{red}{\textless double precision\textgreater} redshift\argin) Return the multiplicative modification to the spectrum.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

stellarPopulationSpectraPostprocessorRecent

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((stellarPopulationSpectraPostprocessorClass))\textgreater
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision multiplier(\textcolor{red}{\textless double precision\textgreater} wavelength\argin,\t
double precision\textgreater} age\argin,\textcolor{red}{\textless double precision\textgreater}
redshift\argin) Return the multiplicative modification to the spectrum.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

stellarPopulationSpectraPostprocessorSequence

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((stellarPopulationSpectraPostprocessorClass))\textgreater
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision multiplier(\textcolor{red}{\textless double precision\textgreater} wavelength\argin,\t
double precision\textgreater} age\argin,\textcolor{red}{\textless double precision\textgreater}
redshift\argin) Return the multiplicative modification to the spectrum.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

stellarPopulationSpectraPostprocessorUnescaped

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((stellarPopulationSpectraPostprocessorClass))\textgreate
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision multiplier(\textcolor{red}{\textless double precision\textgreater} wavelength\argin,\t
double precision\textgreater} age\argin,\textcolor{red}{\textless double precision\textgreater}
redshift\argin) Return the multiplicative modification to the spectrum.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

stellarPopulationStandard

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((stellarPopulationClass))\textgreater} destination\argino Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> interpolate(<type(abundances)> abundances_→, <double> ageMinimum→,<double> ageMaximum→, <type(populationTable)> property↔) Interpolate in the given property to return the mean rate of production of that property from the stellar population between the given minimum and maximum ages.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision rateEnergy(\textcolor{red}{\textless type((abundances))\textgreater} abundances_- \argin,\textcolor{red}{\textless double precision\textgreater} ageMinimum\argin,\textcolor{red}{\te double precision\textgreater} ageMaximum\argin) Return the rate of energy input from this population.

double precision rateRecycling(\textcolor{red}{\textless type((abundances))\textgreater} abundances_ \argin,\textcolor{red}{\textless double precision\textgreater} ageMinimum\argin,\textco double precision\textgreater} ageMaximum\argin) Return the rate of recycling from this population.

double precision rateYield(\textcolor{red}{\textless type((abundances))\textgreater} abundances_- \argin,\textcolor{red}{\textless double precision\textgreater} ageMinimum\argin,\textcolor{red}{\te double precision\textgreater} ageMaximum\argin,\textcolor{red}{\textless integer\textgreater} elementIndex \argin) Return the rate of element yield from this population.

double precision recycledFractionInstantaneous() Return the recycled fraction from this population in the instantaneous approximation.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

class(stellarPopulationSpectraClass) spectra() Return at set of stellar spectra for this population.

<logical> starIsEvolved(<double> massInitial→, <double> metallicity→,<double> age→) Return true if the star of given initial mass and metallicity has evolved off of the main sequence by the given age.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

integer(c_size_t) uniqueID() Return the uniqueID corresponding to this population.

double precision yieldInstantaneous() Return the metal yield from this population in the instantaneous approximation.

stellarSpectraDustAttenuationCalzetti2000

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

double precision attenuation(\textcolor{red}{\textless double precision\textgreater} wavelength\argin,\ double precision\textgreater} age\argin,\textcolor{red}{\textless double precision\textgreater} vBandAttenuation\argin) Return the attenuation, in magnitudes, of stellar spectra due to dust at the given wavelength, age, and V-band extinction.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((stellarSpectraDustAttenuationClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

logical isAgeDependent() Return true if the attenuation may depend on the age of the stellar population.

<logical> isDefault() Return true if this is the default object of this class.

`logical isSeparable()` Return true if the attenuation is separable into a product of functions of wavelength, age, and V-band attenuation.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

stellarSpectraDustAttenuationCardelli1989

`<double> a(<double> x→)` Return fitting function $a(x)$ for the dust attenuation model of Cardelli et al. [1989].

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`double precision attenuation(\textcolor{red}{\textless double precision\textgreater} wavelength\argin,\ double precision\textgreater} age\argin,\textcolor{red}{\textless double precision\textgreater} vBandAttenuation\argin)` Return the attenuation, in magnitudes, of stellar spectra due to dust at the given wavelength, age, and V-band extinction.

`void autoHook()` Insert any event hooks required by this object.

`<double> b(<double> x→)` Return fitting function $b(x)$ for the dust attenuation model of Cardelli et al. [1989].

`void deepCopy(\textcolor{red}{\textless class((stellarSpectraDustAttenuationClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`logical isAgeDependent()` Return true if the attenuation may depend on the age of the stellar population.

`<logical> isDefault()` Return true if this is the default object of this class.

`logical isSeparable()` Return true if the attenuation is separable into a product of functions of wavelength, age, and V-band attenuation.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

stellarSpectraDustAttenuationCharlotFall2000

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

double precision attenuation(\textcolor{red}{\textless double precision\textgreater} wavelength\argin,\ double precision\textgreater} age\argin,\textcolor{red}{\textless double precision\textgreater} vBandAttenuation\argin) Return the attenuation, in magnitudes, of stellar spectra due to dust at the given wavelength, age, and V-band extinction.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((stellarSpectraDustAttenuationClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

logical isAgeDependent() Return true if the attenuation may depend on the age of the stellar population.

<logical> isDefault() Return true if this is the default object of this class.

logical isSeparable() Return true if the attenuation is separable into a product of functions of wavelength, age, and V-band attenuation.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

stellarSpectraDustAttenuationClass

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`double precision attenuation(\textcolor{red}{\textless double precision\textgreater} wavelength\argin,\ double precision\textgreater} age\argin,\textcolor{red}{\textless double precision\textgreater} vBandAttenuation\argin)` Return the attenuation, in magnitudes, of stellar spectra due to dust at the given wavelength, age, and V-band extinction.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((stellarSpectraDustAttenuationClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`logical isAgeDependent()` Return true if the attenuation may depend on the age of the stellar population.

`<logical> isDefault()` Return true if this is the default object of this class.

`logical isSeparable()` Return true if the attenuation is separable into a product of functions of wavelength, age, and V-band attenuation.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

stellarSpectraDustAttenuationGordon2003

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

double precision attenuation(\textcolor{red}{\textless double precision\textgreater} wavelength\argin,\
double precision\textgreater} age\argin,\textcolor{red}{\textless double precision\textgreater}
vBandAttenuation\argin) Return the attenuation, in magnitudes, of stellar spectra due to dust at the given wavelength, age, and V-band extinction.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((stellarSpectraDustAttenuationClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

logical isAgeDependent() Return true if the attenuation may depend on the age of the stellar population.

<logical> isDefault() Return true if this is the default object of this class.

logical isSeparable() Return true if the attenuation is separable into a product of functions of wavelength, age, and V-band attenuation.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

stellarSpectraDustAttenuationPrevotBouchet

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

`double precision attenuation(\textcolor{red}{\textless double precision\textgreater} wavelength\argin,\ double precision\textgreater} age\argin,\textcolor{red}{\textless double precision\textgreater} vBandAttenuation\argin)` Return the attenuation, in magnitudes, of stellar spectra due to dust at the given wavelength, age, and V-band extinction.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((stellarSpectraDustAttenuationClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`logical isAgeDependent()` Return true if the attenuation may depend on the age of the stellar population.

`<logical> isDefault()` Return true if this is the default object of this class.

`logical isSeparable()` Return true if the attenuation is separable into a product of functions of wavelength, age, and V-band attenuation.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

stellarSpectraDustAttenuationTabulated

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`double precision attenuation(\textcolor{red}{\textless double precision\textgreater} wavelength\argin,\ double precision\textgreater} age\argin,\textcolor{red}{\textless double precision\textgreater} vBandAttenuation\argin)` Return the attenuation, in magnitudes, of stellar spectra due to dust at the given wavelength, age, and V-band extinction.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((stellarSpectraDustAttenuationClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`logical isAgeDependent()` Return true if the attenuation may depend on the age of the stellar population.

`<logical> isDefault()` Return true if this is the default object of this class.

`logical isSeparable()` Return true if the attenuation is separable into a product of functions of wavelength, age, and V-band attenuation.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

stellarSpectraDustAttenuationWittGordon2000

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`double precision attenuation(\textcolor{red}{\textless double precision\textgreater} wavelength\argin, double precision\textgreater} age\argin,\textcolor{red}{\textless double precision\textgreater} vBandAttenuation\argin)` Return the attenuation, in magnitudes, of stellar spectra due to dust at the given wavelength, age, and V-band extinction.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((stellarSpectraDustAttenuationClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

logical isAgeDependent() Return true if the attenuation may depend on the age of the stellar population.

<logical> isDefault() Return true if this is the default object of this class.

logical isSeparable() Return true if the attenuation is separable into a product of functions of wavelength, age, and V-band attenuation.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

stellarSpectraDustAttenuationZero

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

double precision attenuation(\textcolor{red}{\textless double precision\textgreater} wavelength\argin,\
double precision\textgreater} age\argin,\textcolor{red}{\textless double precision\textgreater}
vBandAttenuation\argin) Return the attenuation, in magnitudes, of stellar spectra due to dust at the given wavelength, age, and V-band extinction.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((stellarSpectraDustAttenuationClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

logical isAgeDependent() Return true if the attenuation may depend on the age of the stellar population.

<logical> isDefault() Return true if this is the default object of this class.

logical isSeparable() Return true if the attenuation is separable into a product of functions of wavelength, age, and V-band attenuation.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

stellarTracksClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((stellarTracksClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision luminosity(\textcolor{red}{\textless double precision\textgreater} initialMass\argin,\ double precision\textgreater} metallicity\argin,\textcolor{red}{\textless double precision\textgrea age\argin) Returns the bolometric luminosity of a star of given initialMass, metallicity and age.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision temperatureEffective(\textcolor{red}{\textless double precision\textgreater} initialMass\argin,\textcolor{red}{\textless double precision\textgreater} metallicity\argin,\textcc double precision\textgreater} age\argin) Returns the effective temperature of a star of given initialMass, metallicity and age.

stellarTracksFile

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((stellarTracksClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<double> interpolate() <integer(c_size_t)(2)> interpolationIndicesMetallicity→, <integer(c_size_- t)(2,2)> interpolationIndicesMass→, <integer(c_size_t)(2,2,2)> interpolationIndicesAge→, <double(2)> interpolationFactorsMetallicity→, <double(2,2)> interpolationFactorsMass→, <double(2,2,2)> interpolationFactorsAge→, <double(:,:,:)> stellarTracks→

<void> interpolationCompute() <integer(c_size_t)(2)> interpolationIndicesMetallicity←, <integer(c_- size_t)(2,2)> interpolationIndicesMass←, <integer(c_size_t)(2,2,2)> interpolationIndicesAge←, <double(2)> interpolationFactorsMetallicity←, <double(2,2)> interpolationFactorsMass←, <double(2,2,2)> interpolationFactorsAge←, <logical> metallicityOutOfRange←, <logical> massOut- OfRange←, <logical> ageOutOfRange←

<logical> isDefault() Return true if this is the default object of this class.

double precision luminosity(\textcolor{red}{\textless double precision\textgreater} initialMass\argin,\ double precision\textgreater} metallicity\argin,\textcolor{red}{\textless double precision\textgrea age\argin) Returns the bolometric luminosity of a star of given initialMass, metallicity and age.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision temperatureEffective(\textcolor{red}{\textless double precision\textgreater} initialMass\argin,\textcolor{red}{\textless double precision\textgreater} metallicity\argin,\textcc double precision\textgreater} age\argin) Returns the effective temperature of a star of given initialMass, metallicity and age.

stellarWindsClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((stellarWindsClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision rateMassLoss(\textcolor{red}{\textless double precision\textgreater} initialMass\argin double precision\textgreater} age\argin,\textcolor{red}{\textless double precision\textgreater} metallicity\argin) Return the mass loss rate (in $M_\odot$/Gyr) from stars of given initialMass, age and metallicity.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision velocityTerminal(\textcolor{red}{\textless double precision\textgreater} initialMass\argin,\textcolor{red}{\textless double precision\textgreater} age\argin,\textcolor{red}{\textless double precision\textgreater} metallicity\argin) Return the terminal velocity (in km/s) of winds from stars of given initialMass, age and metallicity.

stellarWindsLeitherer1992

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\[1] character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((stellarWindsClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex[1] logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges[1] Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision rateMassLoss(\textcolor{red}{\textless double precision\textgreater} initialMass\argin[1] double precision\textgreater} age\argin,\textcolor{red}{\textless double precision\textgreater} metallicity\argin) Return the mass loss rate (in $M_\odot$/Gyr) from stars of given initialMass, age and metallicity.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textless type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision velocityTerminal(\textcolor{red}{\textless double precision\textgreater} initialMass\argin,\textcolor{red}{\textless double precision\textgreater} age\argin,\textcolor{red}{\textless double precision\textgreater} metallicity\argin) Return the terminal velocity (in km/s) of winds from stars of given initialMass, age and metallicity.

supernovaePopulationIIIClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((supernovaePopulationIIIClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision energyCumulative(\textcolor{red}{\textless double precision\textgreater} initialMass\argin,\textcolor{red}{\textless double precision\textgreater} age\argin,\textcolor{red} double precision\textgreater} metallicity\argin) Return the cumulative energy input from Population III supernovae from stars of given initialMass, age and metallicity.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

supernovaePopulationIIIHegerWoosley2002

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((supernovaePopulationIIIClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

`double precision energyCumulative(\textcolor{red}{\textless double precision\textgreater} initialMass\argin,\textcolor{red}{\textless double precision\textgreater} age\argin,\textcolor{red} double precision\textgreater} metallicity\argin)` Return the cumulative energy input from Population III supernovae from stars of given `initialMass`, `age` and `metallicity`.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

supernovaeTypeIaClass

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((supernovaeTypeIaClass))\textgreater} destination\arginou` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision number(\textcolor{red}{\textless double precision\textgreater} initialMass\argin,\text double precision\textgreater} age\argin,\textcolor{red}{\textless double precision\textgreater} metallicity\argin)` Return the cumulative number of Type Ia supernovae from a stellar population of the given `initialMass`, `age`, and `metallicity`.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision yield(\textcolor{red}{\textless double precision\textgreater} initialMass\argin,\textc double precision\textgreater} age\argin,\textcolor{red}{\textless double precision\textgreater} metallicity\argin,\textcolor{red}{\textless integer\textgreater} atomIndex\argin) Return the cumulative yield from Type Ia supernoave from a stellar population of the given initialMass, age, and metallicity.

supernovaeTypeIaNagashima2005

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((supernovaeTypeIaClass))\textgreater} destination\arginou Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision number(\textcolor{red}{\textless double precision\textgreater} initialMass\argin,\text double precision\textgreater} age\argin,\textcolor{red}{\textless double precision\textgreater} metallicity\argin) Return the cumulative number of Type Ia supernovae from a stellar population of the given initialMass, age, and metallicity.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision yield(\textcolor{red}{\textless double precision\textgreater} initialMass\argin,\textc double precision\textgreater} age\argin,\textcolor{red}{\textless double precision\textgreater} metallicity\argin,\textcolor{red}{\textless integer\textgreater} atomIndex\argin)` Return the cumulative yield from Type Ia supernoave from a stellar population of the given `initialMass`, `age`, and `metallicity`.

surveyGeometryBaldry2012GAMA

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`double precision angularPower(\textcolor{red}{\textless integer\textgreater} i\argin,\textcolor{red}{\t integer\textgreater} j\argin,\textcolor{red}{\textless integer\textgreater} l\argin)` Return C_ℓ^{ij}, where $(2\ell+1)C_\ell^{ij} = \sum_{m=-\ell}^{+\ell} \Psi_{\ell m}^{i} \Psi_{\ell m}^{j*}$, and $\Psi_{\ell m}^{i}$ are the cofficients of the spherical harmonic expansion of the $i^{\rm th}$ field.

`logical angularPowerAvailable()` Returns true if angular power spectrum of survey window function is available.

`integer angularPowerMaximumDegree()` Return the maximum degree, $\ell_{\rm max}$, for which the angular power is available.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((surveyGeometryClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision distanceMaximum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} magnitudeAbsolute\argin,\textcol double precision\textgreater} luminosity\argin,\textcolor{red}{\textless integer\textgreater} field\argin)` Returns the maximum distance (in Mpc) at which a galaxy of the specified `mass` (in $M_\odot$) could be detected.

`double precision distanceMinimum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} magnitudeAbsolute\argin,\textcol double precision\textgreater} luminosity\argin,\textcolor{red}{\textless integer\textgreater} field\argin)` Returns the minimum distance (in Mpc) at which a galaxy of the specified `mass` (in $M_\odot$) would be included in the survey.

`integer fieldCount()` Returns the number of distinct fields included in the survey.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<void> initialize()` Initialize an instance of the MANGLE survey geometry class.

`<logical> isDefault()` Return true if this is the default object of this class.

`<type(varying_string) > mangleDirectory()` Return the directory containing MANGLE files for this survey geometry.

`<type(varying_string)(:)  > mangleFiles()` Return array of MANGLE filenames for this survey geometry.

`type(varying_string) objectType()` Return the type of the object.

`logical pointIncluded(\textcolor{red}{\textless double precision\textgreater} point\argin,\textcolor{re double precision\textgreater} mass\argin)` Return true if the given Cartesian point lies within the survey bounds for the given mass limit.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`double precision solidAngle(\textcolor{red}{\textless integer\textgreater} field\argin)` Return the solid angle (in steradians) of the survey.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision volumeMaximum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless integer\textgreater} field\argin)` Returns the maximum volume (in Mpc^3) at which a galaxy of the specified `mass` (in $M_\odot$) could be detected.

`logical windowFunctionAvailable()` Returns true if survey 3-D window functions are available.

`void windowFunctions(\textcolor{red}{\textless double precision\textgreater} mass1\argin,\textcolor{red double precision\textgreater} mass2\argin,\textcolor{red}{\textless integer\textgreater} gridCount\argin,\textcolor{red}{\textless double precision\textgreater} boxLength\argout,\textcolor complex((c_double_complex))\textgreater} windowFunction1\argout,\textcolor{red}{\textless complex((c_double_complex))\textgreater} windowFunction2\argout)` Returns the window functions on a grid of the specified size (`gridCount` cells in each dimension) for galaxies of the specified `mass1` and `mass2` (in $M_\odot$). The `boxLength` should be set to an appropriate value to fully enclose (with sufficient buffering to allow for Fourier transformation) the two window functions.

surveyGeometryBernardi2013SDSS

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`double precision angularPower(\textcolor{red}{\textless integer\textgreater} i\argin,\textcolor{red}{\t integer\textgreater} j\argin,\textcolor{red}{\textless integer\textgreater} l\argin)` Return C_ℓ^{ij}, where $(2\ell+1)C_\ell^{ij} = \sum_{m=-\ell}^{+\ell} \Psi_{\ell m}^i \Psi_{\ell m}^{j*}$, and $\Psi_{\ell m}^i$ are the cofficients of the spherical harmonic expansion of the $i^{\rm th}$ field.

`logical angularPowerAvailable()` Returns true if angular power spectrum of survey window function is available.

`integer angularPowerMaximumDegree()` Return the maximum degree, $\ell_{\rm max}$, for which the angular power is available.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((surveyGeometryClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision distanceMaximum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} magnitudeAbsolute\argin,\textcol double precision\textgreater} luminosity\argin,\textcolor{red}{\textless integer\textgreater} field\argin)` Returns the maximum distance (in Mpc) at which a galaxy of the specified `mass` (in $M_\odot$) could be detected.

`double precision distanceMinimum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} magnitudeAbsolute\argin,\textcol double precision\textgreater} luminosity\argin,\textcolor{red}{\textless integer\textgreater} field\argin)` Returns the minimum distance (in Mpc) at which a galaxy of the specified `mass` (in $M_\odot$) would be included in the survey.

`integer fieldCount()` Returns the number of distinct fields included in the survey.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<void> initialize()` Initialize an instance of the MANGLE survey geometry class.

`<logical> isDefault()` Return true if this is the default object of this class.

`<type(varying_string) > mangleDirectory()` Return the directory containing MANGLE files for this survey geometry.

`<type(varying_string)(:) > mangleFiles()` Return array of MANGLE filenames for this survey geometry.

`type(varying_string) objectType()` Return the type of the object.

`logical pointIncluded(\textcolor{red}{\textless double precision\textgreater} point\argin,\textcolor{re double precision\textgreater} mass\argin)` Return true if the given Cartesian point lies within the survey bounds for the given mass limit.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

double precision solidAngle(\textcolor{red}{\textless integer\textgreater} field\argin) Return the solid angle (in steradians) of the survey.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision volumeMaximum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless integer\textgreater} field\argin) Returns the maximum volume (in Mpc^3) at which a galaxy of the specified mass (in $M_\odot$) could be detected.

logical windowFunctionAvailable() Returns true if survey 3-D window functions are available.

void windowFunctions(\textcolor{red}{\textless double precision\textgreater} mass1\argin,\textcolor{red} double precision\textgreater} mass2\argin,\textcolor{red}{\textless integer\textgreater} gridCount\argin,\textcolor{red}{\textless double precision\textgreater} boxLength\argout,\textcolor complex((c_double_complex))\textgreater} windowFunction1\argout,\textcolor{red}{\textless complex((c_double_complex))\textgreater} windowFunction2\argout) Returns the window functions on a grid of the specified size (gridCount cells in each dimension) for galaxies of the specified mass1 and mass2 (in $M_\odot$). The boxLength should be set to an appropriate value to fully enclose (with sufficient buffering to allow for Fourier transformation) the two window functions.

surveyGeometryCaputi2011UKIDSSUDS

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

double precision angularPower(\textcolor{red}{\textless integer\textgreater} i\argin,\textcolor{red}{\t integer\textgreater} j\argin,\textcolor{red}{\textless integer\textgreater} l\argin) Return C_ℓ^{ij}, where $(2\ell+1)C_\ell^{ij} = \sum_{m=-\ell}^{+\ell} \Psi_{\ell m}^i \Psi_{\ell m}^{j*}$, and $\Psi_{\ell m}^i$ are the cofficients of the spherical harmonic expansion of the i^{th} field.

logical angularPowerAvailable() Returns true if angular power spectrum of survey window function is available.

integer angularPowerMaximumDegree() Return the maximum degree, ℓ_{max}, for which the angular power is available.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((surveyGeometryClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

`double precision distanceMaximum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} magnitudeAbsolute\argin,\textcol double precision\textgreater} luminosity\argin,\textcolor{red}{\textless integer\textgreater} field\argin)` Returns the maximum distance (in Mpc) at which a galaxy of the specified `mass` (in $M_\odot$) could be detected.

`double precision distanceMinimum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} magnitudeAbsolute\argin,\textcol double precision\textgreater} luminosity\argin,\textcolor{red}{\textless integer\textgreater} field\argin)` Returns the minimum distance (in Mpc) at which a galaxy of the specified `mass` (in $M_\odot$) would be included in the survey.

`integer fieldCount()` Returns the number of distinct fields included in the survey.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`logical pointIncluded(\textcolor{red}{\textless double precision\textgreater} point\argin,\textcolor{re double precision\textgreater} mass\argin)` Return true if the given Cartesian point lies within the survey bounds for the given mass limit.

`<void> randomsInitialize()` Initialize arrays of random points to define the survey angular geometry.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`double precision solidAngle(\textcolor{red}{\textless integer\textgreater} field\argin)` Return the solid angle (in steradians) of the survey.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision volumeMaximum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless integer\textgreater} field\argin)` Returns the maximum volume (in Mpc^3) at which a galaxy of the specified `mass` (in $M_\odot$) could be detected.

`logical windowFunctionAvailable()` Returns true if survey 3-D window functions are available.

`void windowFunctions(\textcolor{red}{\textless double precision\textgreater} mass1\argin,\textcolor{red double precision\textgreater} mass2\argin,\textcolor{red}{\textless integer\textgreater} gridCount\argin,\textcolor{red}{\textless double precision\textgreater} boxLength\argout,\textcolor complex((c_double_complex))\textgreater} windowFunction1\argout,\textcolor{red}{\textless`

complex((c_double_complex))\textgreater} windowFunction2\argout) Returns the window functions on a grid of the specified size (gridCount cells in each dimension) for galaxies of the specified mass1 and mass2 (in $M_\odot$). The boxLength should be set to an appropriate value to fully enclose (with sufficient buffering to allow for Fourier transformation) the two window functions.

surveyGeometryClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

double precision angularPower(\textcolor{red}{\textless integer\textgreater} i\argin,\textcolor{red}{\t
integer\textgreater} j\argin,\textcolor{red}{\textless integer\textgreater} l\argin)
Return C_ℓ^{ij}, where $(2\ell+1)C_\ell^{ij} = \sum_{m=-\ell}^{+\ell} \Psi_{\ell m}^{i} \Psi_{\ell m}^{j*}$, and $\Psi_{\ell m}^{i}$ are the cofficients of the spherical harmonic expansion of the $i^{\rm th}$ field.

logical angularPowerAvailable() Returns true if angular power spectrum of survey window function is available.

integer angularPowerMaximumDegree() Return the maximum degree, $\ell_{\rm max}$, for which the angular power is available.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((surveyGeometryClass))\textgreater} destination\arginout)
Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision distanceMaximum(\textcolor{red}{\textless double precision\textgreater}
mass\argin,\textcolor{red}{\textless double precision\textgreater} magnitudeAbsolute\argin,\textcol
double precision\textgreater} luminosity\argin,\textcolor{red}{\textless integer\textgreater}
field\argin) Returns the maximum distance (in Mpc) at which a galaxy of the specified mass (in $M_\odot$) could be detected.

double precision distanceMinimum(\textcolor{red}{\textless double precision\textgreater}
mass\argin,\textcolor{red}{\textless double precision\textgreater} magnitudeAbsolute\argin,\textcol
double precision\textgreater} luminosity\argin,\textcolor{red}{\textless integer\textgreater}
field\argin) Returns the minimum distance (in Mpc) at which a galaxy of the specified mass (in $M_\odot$) would be included in the survey.

integer fieldCount() Returns the number of distinct fields included in the survey.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

logical pointIncluded(\textcolor{red}{\textless double precision\textgreater} point\argin,\textcolor{re
double precision\textgreater} mass\argin) Return true if the given Cartesian point lies within the survey bounds for the given mass limit.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

double precision solidAngle(\textcolor{red}{\textless integer\textgreater} field\argin) Return the solid angle (in steradians) of the survey.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision volumeMaximum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless integer\textgreater} field\argin) Returns the maximum volume (in Mpc^3) at which a galaxy of the specified mass (in $M_\odot$) could be detected.

logical windowFunctionAvailable() Returns true if survey 3-D window functions are available.

void windowFunctions(\textcolor{red}{\textless double precision\textgreater} mass1\argin,\textcolor{red double precision\textgreater} mass2\argin,\textcolor{red}{\textless integer\textgreater} gridCount\argin,\textcolor{red}{\textless double precision\textgreater} boxLength\argout,\textcolor complex((c_double_complex))\textgreater} windowFunction1\argout,\textcolor{red}{\textless complex((c_double_complex))\textgreater} windowFunction2\argout) Returns the window functions on a grid of the specified size (gridCount cells in each dimension) for galaxies of the specified mass1 and mass2 (in $M_\odot$). The boxLength should be set to an appropriate value to fully enclose (with sufficient buffering to allow for Fourier transformation) the two window functions.

surveyGeometryCombined

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

double precision angularPower(\textcolor{red}{\textless integer\textgreater} i\argin,\textcolor{red}{\t integer\textgreater} j\argin,\textcolor{red}{\textless integer\textgreater} l\argin) Return C_ℓ^{ij}, where $(2\ell+1)C_\ell^{ij} = \sum_{m=-\ell}^{+\ell} \Psi_{\ell m}^i \Psi_{\ell m}^{j*}$, and $\Psi_{\ell m}^i$ are the cofficients of the spherical harmonic expansion of the i^{th} field.

logical angularPowerAvailable() Returns true if angular power spectrum of survey window function is available.

integer angularPowerMaximumDegree() Return the maximum degree, ℓ_{max}, for which the angular power is available.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((surveyGeometryClass))\textgreater} destination\arginout) Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision distanceMaximum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} magnitudeAbsolute\argin,\textcol double precision\textgreater} luminosity\argin,\textcolor{red}{\textless integer\textgreater} field\argin)` Returns the maximum distance (in Mpc) at which a galaxy of the specified `mass` (in $M_\odot$) could be detected.

`double precision distanceMinimum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} magnitudeAbsolute\argin,\textcol double precision\textgreater} luminosity\argin,\textcolor{red}{\textless integer\textgreater} field\argin)` Returns the minimum distance (in Mpc) at which a galaxy of the specified `mass` (in $M_\odot$) would be included in the survey.

`integer fieldCount()` Returns the number of distinct fields included in the survey.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`logical pointIncluded(\textcolor{red}{\textless double precision\textgreater} point\argin,\textcolor{re double precision\textgreater} mass\argin)` Return true if the given Cartesian point lies within the survey bounds for the given mass limit.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`double precision solidAngle(\textcolor{red}{\textless integer\textgreater} field\argin)` Return the solid angle (in steradians) of the survey.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision volumeMaximum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless integer\textgreater} field\argin)` Returns the maximum volume (in Mpc^3) at which a galaxy of the specified `mass` (in $M_\odot$) could be detected.

`logical windowFunctionAvailable()` Returns true if survey 3-D window functions are available.

`void windowFunctions(\textcolor{red}{\textless double precision\textgreater} mass1\argin,\textcolor{red}{\textless double precision\textgreater} mass2\argin,\textcolor{red}{\textless integer\textgreater} gridCount\argin,\textcolor{red}{\textless double precision\textgreater} boxLength\argout,\textcolor{red}{\textless complex((c_double_complex))\textgreater} windowFunction1\argout,\textcolor{red}{\textless complex((c_double_complex))\textgreater} windowFunction2\argout)` Returns the window functions on a grid of the specified size (`gridCount` cells in each dimension) for galaxies of the specified `mass1` and `mass2` (in $M_\odot$). The `boxLength` should be set to an appropriate value to fully enclose (with sufficient buffering to allow for Fourier transformation) the two window functions.

surveyGeometryDavidzon2013VIPERS

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\[\], \textcolor{red}{\textless character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`double precision angularPower(\textcolor{red}{\textless integer\textgreater} i\argin,\textcolor{red}{\textless integer\textgreater} j\argin,\textcolor{red}{\textless integer\textgreater} l\argin)` Return C_ℓ^{ij}, where $(2\ell+1)C_\ell^{ij} = \sum_{m=-\ell}^{+\ell} \Psi_{\ell m}^i \Psi_{\ell m}^{j*}$, and $\Psi_{\ell m}^i$ are the cofficients of the spherical harmonic expansion of the $i^{\rm th}$ field.

`logical angularPowerAvailable()` Returns true if angular power spectrum of survey window function is available.

`integer angularPowerMaximumDegree()` Return the maximum degree, $\ell_{\rm max}$, for which the angular power is available.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((surveyGeometryClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\textcolor{red}{\textless logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision distanceMaximum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} magnitudeAbsolute\argin,\textcolor{red}{\textless double precision\textgreater} luminosity\argin,\textcolor{red}{\textless integer\textgreater} field\argin)` Returns the maximum distance (in Mpc) at which a galaxy of the specified `mass` (in $M_\odot$) could be detected.

`double precision distanceMinimum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} magnitudeAbsolute\argin,\textcolor{red}{\textless double precision\textgreater} luminosity\argin,\textcolor{red}{\textless integer\textgreater} field\argin)` Returns the minimum distance (in Mpc) at which a galaxy of the specified `mass` (in $M_\odot$) would be included in the survey.

`integer fieldCount()` Returns the number of distinct fields included in the survey.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDigest\argin)` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<void> initialize()` Initialize an instance of the MANGLE survey geometry class.

`<logical>` `isDefault()` Return true if this is the default object of this class.

`<type(varying_string) >` `mangleDirectory()` Return the directory containing MANGLE files for this survey geometry.

`<type(varying_string)(:)  >` `mangleFiles()` Return array of MANGLE filenames for this survey geometry.

`type(varying_string) objectType()` Return the type of the object.

`logical pointIncluded(\textcolor{red}{\textless double precision\textgreater} point\argin,\textcolor{re double precision\textgreater} mass\argin)` Return true if the given Cartesian point lies within the survey bounds for the given mass limit.

`<integer>` `referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void>` `referenceCountIncrement()` Increment the reference count to this object.

`<void>` `referenceCountReset()` Reset the reference count to this object to 0.

`double precision solidAngle(\textcolor{red}{\textless integer\textgreater} field\argin)` Return the solid angle (in steradians) of the survey.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision volumeMaximum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless integer\textgreater} field\argin)` Returns the maximum volume (in Mpc^3) at which a galaxy of the specified `mass` (in $M_\odot$) could be detected.

`logical windowFunctionAvailable()` Returns true if survey 3-D window functions are available.

`void windowFunctions(\textcolor{red}{\textless double precision\textgreater} mass1\argin,\textcolor{red double precision\textgreater} mass2\argin,\textcolor{red}{\textless integer\textgreater} gridCount\argin,\textcolor{red}{\textless double precision\textgreater} boxLength\argout,\textcolor complex((c_double_complex))\textgreater} windowFunction1\argout,\textcolor{red}{\textless complex((c_double_complex))\textgreater} windowFunction2\argout)` Returns the window functions on a grid of the specified size (`gridCount` cells in each dimension) for galaxies of the specified `mass1` and `mass2` (in $M_\odot$). The `boxLength` should be set to an appropriate value to fully enclose (with sufficient buffering to allow for Fourier transformation) the two window functions.

surveyGeometryFullSky

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`double precision angularPower(\textcolor{red}{\textless integer\textgreater} i\argin,\textcolor{red}{\t integer\textgreater} j\argin,\textcolor{red}{\textless integer\textgreater} l\argin)` Return C_ℓ^{ij}, where $(2\ell+1)C_\ell^{ij} = \sum_{m=-\ell}^{+\ell} \Psi_{\ell m}^i \Psi_{\ell m}^{j*}$, and $\Psi_{\ell m}^i$ are the cofficients of the spherical harmonic expansion of the $i^{\rm th}$ field.

`logical angularPowerAvailable()` Returns true if angular power spectrum of survey window function is available.

`integer angularPowerMaximumDegree()` Return the maximum degree, $\ell_{\rm max}$, for which the angular power is available.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((surveyGeometryClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision distanceMaximum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} magnitudeAbsolute\argin,\textcol double precision\textgreater} luminosity\argin,\textcolor{red}{\textless integer\textgreater} field\argin)` Returns the maximum distance (in Mpc) at which a galaxy of the specified `mass` (in $M_\odot$) could be detected.

`double precision distanceMinimum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} magnitudeAbsolute\argin,\textcol double precision\textgreater} luminosity\argin,\textcolor{red}{\textless integer\textgreater} field\argin)` Returns the minimum distance (in Mpc) at which a galaxy of the specified `mass` (in $M_\odot$) would be included in the survey.

`integer fieldCount()` Returns the number of distinct fields included in the survey.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`logical pointIncluded(\textcolor{red}{\textless double precision\textgreater} point\argin,\textcolor{re double precision\textgreater} mass\argin)` Return true if the given Cartesian point lies within the survey bounds for the given mass limit.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`double precision solidAngle(\textcolor{red}{\textless integer\textgreater} field\argin)` Return the solid angle (in steradians) of the survey.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision volumeMaximum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless integer\textgreater} field\argin)` Returns the maximum volume (in Mpc^3) at which a galaxy of the specified `mass` (in $M_\odot$) could be detected.

`logical windowFunctionAvailable()` Returns true if survey 3-D window functions are available.

`void windowFunctions(\textcolor{red}{\textless double precision\textgreater} mass1\argin,\textcolor{red} double precision\textgreater} mass2\argin,\textcolor{red}{\textless integer\textgreater} gridCount\argin,\textcolor{red}{\textless double precision\textgreater} boxLength\argout,\textcolor complex((c_double_complex))\textgreater} windowFunction1\argout,\textcolor{red}{\textless complex((c_double_complex))\textgreater} windowFunction2\argout)` Returns the window functions on a grid of the specified size (`gridCount` cells in each dimension) for galaxies of the specified `mass1` and `mass2` (in $M_\odot$). The `boxLength` should be set to an appropriate value to fully enclose (with sufficient buffering to allow for Fourier transformation) the two window functions.

surveyGeometryGunawardhana2013SDSS

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`double precision angularPower(\textcolor{red}{\textless integer\textgreater} i\argin,\textcolor{red}{\t integer\textgreater} j\argin,\textcolor{red}{\textless integer\textgreater} l\argin)` Return C_ℓ^{ij}, where $(2\ell+1)C_\ell^{ij} = \sum_{m=-\ell}^{+\ell} \Psi_{\ell m}^i \Psi_{\ell m}^{j*}$, and $\Psi_{\ell m}^i$ are the cofficients of the spherical harmonic expansion of the i^{th} field.

`logical angularPowerAvailable()` Returns true if angular power spectrum of survey window function is available.

`integer angularPowerMaximumDegree()` Return the maximum degree, ℓ_{max}, for which the angular power is available.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((surveyGeometryClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision distanceMaximum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} magnitudeAbsolute\argin,\textcol double precision\textgreater} luminosity\argin,\textcolor{red}{\textless integer\textgreater} field\argin)` Returns the maximum distance (in Mpc) at which a galaxy of the specified `mass` (in $M_\odot$) could be detected.

`double precision distanceMinimum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} magnitudeAbsolute\argin,\textcol double precision\textgreater} luminosity\argin,\textcolor{red}{\textless integer\textgreater} field\argin)` Returns the minimum distance (in Mpc) at which a galaxy of the specified `mass` (in $M_\odot$) would be included in the survey.

`integer fieldCount()` Returns the number of distinct fields included in the survey.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<void> initialize()` Initialize an instance of the MANGLE survey geometry class.

`<logical> isDefault()` Return true if this is the default object of this class.

`<type(varying_string) > mangleDirectory()` Return the directory containing MANGLE files for this survey geometry.

`<type(varying_string)(:) > mangleFiles()` Return array of MANGLE filenames for this survey geometry.

`type(varying_string) objectType()` Return the type of the object.

`logical pointIncluded(\textcolor{red}{\textless double precision\textgreater} point\argin,\textcolor{re double precision\textgreater} mass\argin)` Return true if the given Cartesian point lies within the survey bounds for the given mass limit.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`double precision solidAngle(\textcolor{red}{\textless integer\textgreater} field\argin)` Return the solid angle (in steradians) of the survey.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision volumeMaximum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless integer\textgreater} field\argin)` Returns the maximum volume (in Mpc^3) at which a galaxy of the specified `mass` (in $M_\odot$) could be detected.

`logical windowFunctionAvailable()` Returns true if survey 3-D window functions are available.

`void windowFunctions(\textcolor{red}{\textless double precision\textgreater} mass1\argin,\textcolor{red} double precision\textgreater} mass2\argin,\textcolor{red}{\textless integer\textgreater} gridCount\argin,\textcolor{red}{\textless double precision\textgreater} boxLength\argout,\textcolor complex((c_double_complex))\textgreater} windowFunction1\argout,\textcolor{red}{\textless complex((c_double_complex))\textgreater} windowFunction2\argout)` Returns the window

functions on a grid of the specified size (`gridCount` cells in each dimension) for galaxies of the specified `mass1` and `mass2` (in $M_\odot$). The `boxLength` should be set to an appropriate value to fully enclose (with sufficient buffering to allow for Fourier transformation) the two window functions.

surveyGeometryHearin2014SDSS

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`double precision angularPower(\textcolor{red}{\textless integer\textgreater} i\argin,\textcolor{red}{\t integer\textgreater} j\argin,\textcolor{red}{\textless integer\textgreater} l\argin)` Return C_ℓ^{ij}, where $(2\ell+1)C_\ell^{ij} = \sum_{m=-\ell}^{+\ell} \Psi_{\ell m}^i \Psi_{\ell m}^{j*}$, and $\Psi_{\ell m}^i$ are the cofficients of the spherical harmonic expansion of the $i^{\rm th}$ field.

`logical angularPowerAvailable()` Returns true if angular power spectrum of survey window function is available.

`integer angularPowerMaximumDegree()` Return the maximum degree, $\ell_{\rm max}$, for which the angular power is available.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((surveyGeometryClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision distanceMaximum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} magnitudeAbsolute\argin,\textcol double precision\textgreater} luminosity\argin,\textcolor{red}{\textless integer\textgreater} field\argin)` Returns the maximum distance (in Mpc) at which a galaxy of the specified `mass` (in $M_\odot$) could be detected.

`double precision distanceMinimum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} magnitudeAbsolute\argin,\textcol double precision\textgreater} luminosity\argin,\textcolor{red}{\textless integer\textgreater} field\argin)` Returns the minimum distance (in Mpc) at which a galaxy of the specified `mass` (in $M_\odot$) would be included in the survey.

`integer fieldCount()` Returns the number of distinct fields included in the survey.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<void> initialize()` Initialize an instance of the MANGLE survey geometry class.

`<logical> isDefault()` Return true if this is the default object of this class.

`<type(varying_string) > mangleDirectory()` Return the directory containing MANGLE files for this survey geometry.

<type(varying_string)(:) > mangleFiles() Return array of MANGLE filenames for this survey geometry.

type(varying_string) objectType() Return the type of the object.

logical pointIncluded(\textcolor{red}{\textless double precision\textgreater} point\argin,\textcolor{re
double precision\textgreater} mass\argin) Return true if the given Cartesian point lies within the survey bounds for the given mass limit.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

double precision solidAngle(\textcolor{red}{\textless integer\textgreater} field\argin) Return the solid angle (in steradians) of the survey.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision volumeMaximum(\textcolor{red}{\textless double precision\textgreater}
mass\argin,\textcolor{red}{\textless integer\textgreater} field\argin) Returns the maximum volume (in Mpc^3) at which a galaxy of the specified mass (in $M_\odot$) could be detected.

logical windowFunctionAvailable() Returns true if survey 3-D window functions are available.

void windowFunctions(\textcolor{red}{\textless double precision\textgreater} mass1\argin,\textcolor{red
double precision\textgreater} mass2\argin,\textcolor{red}{\textless integer\textgreater}
gridCount\argin,\textcolor{red}{\textless double precision\textgreater} boxLength\argout,\textcolor
complex((c_double_complex))\textgreater} windowFunction1\argout,\textcolor{red}{\textless
complex((c_double_complex))\textgreater} windowFunction2\argout) Returns the window functions on a grid of the specified size (gridCount cells in each dimension) for galaxies of the specified mass1 and mass2 (in $M_\odot$). The boxLength should be set to an appropriate value to fully enclose (with sufficient buffering to allow for Fourier transformation) the two window functions.

surveyGeometryKelvin2014GAMAnear

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

double precision angularPower(\textcolor{red}{\textless integer\textgreater} i\argin,\textcolor{red}{\t
integer\textgreater} j\argin,\textcolor{red}{\textless integer\textgreater} l\argin) Return C_ℓ^{ij}, where $(2\ell+1)C_\ell^{ij} = \sum_{m=-\ell}^{+\ell} \Psi_{\ell m}^i \Psi_{\ell m}^{j*}$, and $\Psi_{\ell m}^i$ are the cofficients of the spherical harmonic expansion of the i^{th} field.

logical angularPowerAvailable() Returns true if angular power spectrum of survey window function is available.

`integer angularPowerMaximumDegree()` Return the maximum degree, $\ell_{\rm max}$, for which the angular power is available.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((surveyGeometryClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision distanceMaximum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} magnitudeAbsolute\argin,\textcol double precision\textgreater} luminosity\argin,\textcolor{red}{\textless integer\textgreater} field\argin)` Returns the maximum distance (in Mpc) at which a galaxy of the specified `mass` (in $M_\odot$) could be detected.

`double precision distanceMinimum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} magnitudeAbsolute\argin,\textcol double precision\textgreater} luminosity\argin,\textcolor{red}{\textless integer\textgreater} field\argin)` Returns the minimum distance (in Mpc) at which a galaxy of the specified `mass` (in $M_\odot$) would be included in the survey.

`integer fieldCount()` Returns the number of distinct fields included in the survey.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<void> initialize()` Initialize an instance of the MANGLE survey geometry class.

`<logical> isDefault()` Return true if this is the default object of this class.

`<type(varying_string) > mangleDirectory()` Return the directory containing MANGLE files for this survey geometry.

`<type(varying_string)(:)  > mangleFiles()` Return array of MANGLE filenames for this survey geometry.

`type(varying_string) objectType()` Return the type of the object.

`logical pointIncluded(\textcolor{red}{\textless double precision\textgreater} point\argin,\textcolor{re double precision\textgreater} mass\argin)` Return true if the given Cartesian point lies within the survey bounds for the given mass limit.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`double precision solidAngle(\textcolor{red}{\textless integer\textgreater} field\argin)` Return the solid angle (in steradians) of the survey.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision volumeMaximum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless integer\textgreater} field\argin)` Returns the maximum volume (in Mpc^3) at which a galaxy of the specified `mass` (in $M_\odot$) could be detected.

`logical windowFunctionAvailable()` Returns true if survey 3-D window functions are available.

`void windowFunctions(\textcolor{red}{\textless double precision\textgreater} mass1\argin,\textcolor{red} double precision\textgreater} mass2\argin,\textcolor{red}{\textless integer\textgreater} gridCount\argin,\textcolor{red}{\textless double precision\textgreater} boxLength\argout,\textcolor complex((c_double_complex))\textgreater} windowFunction1\argout,\textcolor{red}{\textless complex((c_double_complex))\textgreater} windowFunction2\argout)` Returns the window functions on a grid of the specified size (`gridCount` cells in each dimension) for galaxies of the specified `mass1` and `mass2` (in $M_\odot$). The `boxLength` should be set to an appropriate value to fully enclose (with sufficient buffering to allow for Fourier transformation) the two window functions.

surveyGeometryLiWhite2009SDSS

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`double precision angularPower(\textcolor{red}{\textless integer\textgreater} i\argin,\textcolor{red}{\t integer\textgreater} j\argin,\textcolor{red}{\textless integer\textgreater} l\argin)` Return C_ℓ^{ij}, where $(2\ell+1)C_\ell^{ij} = \sum_{m=-\ell}^{+\ell} \Psi_{\ell m}^i \Psi_{\ell m}^{j*}$, and $\Psi_{\ell m}^i$ are the cofficients of the spherical harmonic expansion of the i^{th} field.

`logical angularPowerAvailable()` Returns true if angular power spectrum of survey window function is available.

`integer angularPowerMaximumDegree()` Return the maximum degree, ℓ_{max}, for which the angular power is available.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((surveyGeometryClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision distanceMaximum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} magnitudeAbsolute\argin,\textcol double precision\textgreater} luminosity\argin,\textcolor{red}{\textless integer\textgreater} field\argin)` Returns the maximum distance (in Mpc) at which a galaxy of the specified `mass` (in $M_\odot$) could be detected.

`double precision distanceMinimum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} magnitudeAbsolute\argin,\textcol double precision\textgreater} luminosity\argin,\textcolor{red}{\textless integer\textgreater} field\argin)` Returns the minimum distance (in Mpc) at which a galaxy of the specified `mass` (in $M_\odot$) would be included in the survey.

`integer fieldCount()` Returns the number of distinct fields included in the survey.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`logical pointIncluded(\textcolor{red}{\textless double precision\textgreater} point\argin,\textcolor{re double precision\textgreater} mass\argin)` Return true if the given Cartesian point lies within the survey bounds for the given mass limit.

`<void> randomsInitialize()` Initialize arrays of random points to define the survey angular geometry.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`double precision solidAngle(\textcolor{red}{\textless integer\textgreater} field\argin)` Return the solid angle (in steradians) of the survey.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision volumeMaximum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless integer\textgreater} field\argin)` Returns the maximum volume (in Mpc^3) at which a galaxy of the specified `mass` (in $M_\odot$) could be detected.

`logical windowFunctionAvailable()` Returns true if survey 3-D window functions are available.

`void windowFunctions(\textcolor{red}{\textless double precision\textgreater} mass1\argin,\textcolor{red double precision\textgreater} mass2\argin,\textcolor{red}{\textless integer\textgreater} gridCount\argin,\textcolor{red}{\textless double precision\textgreater} boxLength\argout,\textcolor complex((c_double_complex))\textgreater} windowFunction1\argout,\textcolor{red}{\textless complex((c_double_complex))\textgreater} windowFunction2\argout)` Returns the window functions on a grid of the specified size (`gridCount` cells in each dimension) for galaxies of the specified `mass1` and `mass2` (in $M_\odot$). The `boxLength` should be set to an appropriate value to fully enclose (with sufficient buffering to allow for Fourier transformation) the two window functions.

surveyGeometryLocalGroupClassical

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`double precision angularPower(\textcolor{red}{\textless integer\textgreater} i\argin,\textcolor{red}{\t integer\textgreater} j\argin,\textcolor{red}{\textless integer\textgreater} l\argin)` Return C_ℓ^{ij}, where $(2\ell+1)C_\ell^{ij} = \sum_{m=-\ell}^{+\ell} \Psi_{\ell m}^i \Psi_{\ell m}^{j*}$, and $\Psi_{\ell m}^i$ are the cofficients of the spherical harmonic expansion of the $i^{\rm th}$ field.

`logical angularPowerAvailable()` Returns true if angular power spectrum of survey window function is available.

`integer angularPowerMaximumDegree()` Return the maximum degree, $\ell_{\rm max}$, for which the angular power is available.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((surveyGeometryClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision distanceMaximum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} magnitudeAbsolute\argin,\textcol double precision\textgreater} luminosity\argin,\textcolor{red}{\textless integer\textgreater} field\argin)` Returns the maximum distance (in Mpc) at which a galaxy of the specified `mass` (in $M_\odot$) could be detected.

`double precision distanceMinimum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} magnitudeAbsolute\argin,\textcol double precision\textgreater} luminosity\argin,\textcolor{red}{\textless integer\textgreater} field\argin)` Returns the minimum distance (in Mpc) at which a galaxy of the specified `mass` (in $M_\odot$) would be included in the survey.

`integer fieldCount()` Returns the number of distinct fields included in the survey.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<void> initialize()` Initialize an instance of the MANGLE survey geometry class.

`<logical> isDefault()` Return true if this is the default object of this class.

`<type(varying_string) > mangleDirectory()` Return the directory containing MANGLE files for this survey geometry.

`<type(varying_string)(:) > mangleFiles()` Return array of MANGLE filenames for this survey geometry.

`type(varying_string) objectType()` Return the type of the object.

`logical pointIncluded(\textcolor{red}{\textless double precision\textgreater} point\argin,\textcolor{re double precision\textgreater} mass\argin)` Return true if the given Cartesian point lies within the survey bounds for the given mass limit.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`double precision solidAngle(\textcolor{red}{\textless integer\textgreater} field\argin)` Return the solid angle (in steradians) of the survey.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision volumeMaximum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless integer\textgreater} field\argin)` Returns the maximum volume (in Mpc^3) at which a galaxy of the specified `mass` (in $M_\odot$) could be detected.

`logical windowFunctionAvailable()` Returns true if survey 3-D window functions are available.

`void windowFunctions(\textcolor{red}{\textless double precision\textgreater} mass1\argin,\textcolor{red double precision\textgreater} mass2\argin,\textcolor{red}{\textless integer\textgreater} gridCount\argin,\textcolor{red}{\textless double precision\textgreater} boxLength\argout,\textcolor complex((c_double_complex))\textgreater} windowFunction1\argout,\textcolor{red}{\textless complex((c_double_complex))\textgreater} windowFunction2\argout)` Returns the window functions on a grid of the specified size (`gridCount` cells in each dimension) for galaxies of the specified `mass1` and `mass2` (in $M_\odot$). The `boxLength` should be set to an appropriate value to fully enclose (with sufficient buffering to allow for Fourier transformation) the two window functions.

surveyGeometryLocalGroupDES

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`double precision angularPower(\textcolor{red}{\textless integer\textgreater} i\argin,\textcolor{red}{\t integer\textgreater} j\argin,\textcolor{red}{\textless integer\textgreater} l\argin)` Return C_ℓ^{ij}, where $(2\ell+1)C_\ell^{ij} = \sum_{m=-\ell}^{+\ell} \Psi_{\ell m}^i \Psi_{\ell m}^{j*}$, and $\Psi_{\ell m}^i$ are the cofficients of the spherical harmonic expansion of the i^{th} field.

`logical angularPowerAvailable()` Returns true if angular power spectrum of survey window function is available.

`integer angularPowerMaximumDegree()` Return the maximum degree, ℓ_{max}, for which the angular power is available.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((surveyGeometryClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision distanceMaximum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} magnitudeAbsolute\argin,\textcol double precision\textgreater} luminosity\argin,\textcolor{red}{\textless integer\textgreater} field\argin)` Returns the maximum distance (in Mpc) at which a galaxy of the specified `mass` (in $M_\odot$) could be detected.

`double precision distanceMinimum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} magnitudeAbsolute\argin,\textcol double precision\textgreater} luminosity\argin,\textcolor{red}{\textless integer\textgreater} field\argin)` Returns the minimum distance (in Mpc) at which a galaxy of the specified `mass` (in $M_\odot$) would be included in the survey.

`integer fieldCount()` Returns the number of distinct fields included in the survey.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<void> initialize()` Initialize an instance of the MANGLE survey geometry class.

`<logical> isDefault()` Return true if this is the default object of this class.

`<type(varying_string) > mangleDirectory()` Return the directory containing MANGLE files for this survey geometry.

`<type(varying_string)(:) > mangleFiles()` Return array of MANGLE filenames for this survey geometry.

`type(varying_string) objectType()` Return the type of the object.

`logical pointIncluded(\textcolor{red}{\textless double precision\textgreater} point\argin,\textcolor{re double precision\textgreater} mass\argin)` Return true if the given Cartesian point lies within the survey bounds for the given mass limit.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`double precision solidAngle(\textcolor{red}{\textless integer\textgreater} field\argin)` Return the solid angle (in steradians) of the survey.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision volumeMaximum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless integer\textgreater} field\argin)` Returns the maximum volume (in Mpc^3) at which a galaxy of the specified `mass` (in $M_\odot$) could be detected.

`logical windowFunctionAvailable()` Returns true if survey 3-D window functions are available.

`void windowFunctions(\textcolor{red}{\textless double precision\textgreater} mass1\argin,\textcolor{red} double precision\textgreater} mass2\argin,\textcolor{red}{\textless integer\textgreater} gridCount\argin,\textcolor{red}{\textless double precision\textgreater} boxLength\argout,\textcolor complex((c_double_complex))\textgreater} windowFunction1\argout,\textcolor{red}{\textless complex((c_double_complex))\textgreater} windowFunction2\argout)` Returns the window functions on a grid of the specified size (`gridCount` cells in each dimension) for galaxies of the specified `mass1` and `mass2` (in $M_\odot$). The `boxLength` should be set to an appropriate value to fully enclose (with sufficient buffering to allow for Fourier transformation) the two window functions.

surveyGeometryLocalGroupSDSS

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`double precision angularPower(\textcolor{red}{\textless integer\textgreater} i\argin,\textcolor{red}{\t integer\textgreater} j\argin,\textcolor{red}{\textless integer\textgreater} l\argin)` Return C_ℓ^{ij}, where $(2\ell+1)C_\ell^{ij} = \sum_{m=-\ell}^{+\ell} \Psi_{\ell m}^i \Psi_{\ell m}^{j*}$, and $\Psi_{\ell m}^i$ are the cofficients of the spherical harmonic expansion of the i^{th} field.

`logical angularPowerAvailable()` Returns true if angular power spectrum of survey window function is available.

`integer angularPowerMaximumDegree()` Return the maximum degree, ℓ_{max}, for which the angular power is available.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((surveyGeometryClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision distanceMaximum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} magnitudeAbsolute\argin,\textcol double precision\textgreater} luminosity\argin,\textcolor{red}{\textless integer\textgreater} field\argin)` Returns the maximum distance (in Mpc) at which a galaxy of the specified `mass` (in $M_\odot$) could be detected.

`double precision distanceMinimum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} magnitudeAbsolute\argin,\textcol double precision\textgreater} luminosity\argin,\textcolor{red}{\textless integer\textgreater}`

field\argin) Returns the minimum distance (in Mpc) at which a galaxy of the specified mass (in $M_\odot$) would be included in the survey.

integer fieldCount() Returns the number of distinct fields included in the survey.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<void> initialize() Initialize an instance of the MANGLE survey geometry class.

<logical> isDefault() Return true if this is the default object of this class.

<type(varying_string) > mangleDirectory() Return the directory containing MANGLE files for this survey geometry.

<type(varying_string)(:) > mangleFiles() Return array of MANGLE filenames for this survey geometry.

type(varying_string) objectType() Return the type of the object.

logical pointIncluded(\textcolor{red}{\textless double precision\textgreater} point\argin,\textcolor{re double precision\textgreater} mass\argin) Return true if the given Cartesian point lies within the survey bounds for the given mass limit.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

double precision solidAngle(\textcolor{red}{\textless integer\textgreater} field\argin) Return the solid angle (in steradians) of the survey.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision volumeMaximum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless integer\textgreater} field\argin) Returns the maximum volume (in Mpc^3) at which a galaxy of the specified mass (in $M_\odot$) could be detected.

logical windowFunctionAvailable() Returns true if survey 3-D window functions are available.

void windowFunctions(\textcolor{red}{\textless double precision\textgreater} mass1\argin,\textcolor{red double precision\textgreater} mass2\argin,\textcolor{red}{\textless integer\textgreater} gridCount\argin,\textcolor{red}{\textless double precision\textgreater} boxLength\argout,\textcolor complex((c_double_complex))\textgreater} windowFunction1\argout,\textcolor{red}{\textless complex((c_double_complex))\textgreater} windowFunction2\argout) Returns the window functions on a grid of the specified size (gridCount cells in each dimension) for galaxies of the specified mass1 and mass2 (in $M_\odot$). The boxLength should be set to an appropriate value to fully enclose (with sufficient buffering to allow for Fourier transformation) the two window functions.

surveyGeometryMangle

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`double precision angularPower(\textcolor{red}{\textless integer\textgreater} i\argin,\textcolor{red}{\t integer\textgreater} j\argin,\textcolor{red}{\textless integer\textgreater} l\argin)` Return C_ℓ^{ij}, where $(2\ell+1)C_\ell^{ij} = \sum_{m=-\ell}^{+\ell} \Psi_{\ell m}^i \Psi_{\ell m}^{j*}$, and $\Psi_{\ell m}^i$ are the cofficients of the spherical harmonic expansion of the $i^{\rm th}$ field.

`logical angularPowerAvailable()` Returns true if angular power spectrum of survey window function is available.

`integer angularPowerMaximumDegree()` Return the maximum degree, $\ell_{\rm max}$, for which the angular power is available.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((surveyGeometryClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision distanceMaximum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} magnitudeAbsolute\argin,\textcol double precision\textgreater} luminosity\argin,\textcolor{red}{\textless integer\textgreater} field\argin)` Returns the maximum distance (in Mpc) at which a galaxy of the specified `mass` (in $M_\odot$) could be detected.

`double precision distanceMinimum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} magnitudeAbsolute\argin,\textcol double precision\textgreater} luminosity\argin,\textcolor{red}{\textless integer\textgreater} field\argin)` Returns the minimum distance (in Mpc) at which a galaxy of the specified `mass` (in $M_\odot$) would be included in the survey.

`integer fieldCount()` Returns the number of distinct fields included in the survey.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<void> initialize()` Initialize an instance of the MANGLE survey geometry class.

`<logical> isDefault()` Return true if this is the default object of this class.

`<type(varying_string) > mangleDirectory()` Return the directory containing MANGLE files for this survey geometry.

`<type(varying_string)(:) > mangleFiles()` Return array of MANGLE filenames for this survey geometry.

`type(varying_string) objectType()` Return the type of the object.

`logical pointIncluded(\textcolor{red}{\textless double precision\textgreater} point\argin,\textcolor{re double precision\textgreater} mass\argin)` Return true if the given Cartesian point lies within the survey bounds for the given mass limit.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`double precision solidAngle(\textcolor{red}{\textless integer\textgreater} field\argin)` Return the solid angle (in steradians) of the survey.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision volumeMaximum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless integer\textgreater} field\argin)` Returns the maximum volume (in Mpc^3) at which a galaxy of the specified `mass` (in $M_\odot$) could be detected.

`logical windowFunctionAvailable()` Returns true if survey 3-D window functions are available.

`void windowFunctions(\textcolor{red}{\textless double precision\textgreater} mass1\argin,\textcolor{red double precision\textgreater} mass2\argin,\textcolor{red}{\textless integer\textgreater} gridCount\argin,\textcolor{red}{\textless double precision\textgreater} boxLength\argout,\textcolor complex((c_double_complex))\textgreater} windowFunction1\argout,\textcolor{red}{\textless complex((c_double_complex))\textgreater} windowFunction2\argout)` Returns the window functions on a grid of the specified size (`gridCount` cells in each dimension) for galaxies of the specified `mass1` and `mass2` (in $M_\odot$). The `boxLength` should be set to an appropriate value to fully enclose (with sufficient buffering to allow for Fourier transformation) the two window functions.

surveyGeometryMartin2010ALFALFA

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`double precision angularPower(\textcolor{red}{\textless integer\textgreater} i\argin,\textcolor{red}{\t integer\textgreater} j\argin,\textcolor{red}{\textless integer\textgreater} l\argin)` Return C_ℓ^{ij}, where $(2\ell+1)C_\ell^{ij} = \sum_{m=-\ell}^{+\ell} \Psi_{\ell m}^i \Psi_{\ell m}^{j*}$, and $\Psi_{\ell m}^i$ are the cofficients of the spherical harmonic expansion of the i^{th} field.

`logical angularPowerAvailable()` Returns true if angular power spectrum of survey window function is available.

`integer angularPowerMaximumDegree()` Return the maximum degree, ℓ_{max}, for which the angular power is available.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((surveyGeometryClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision distanceMaximum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} magnitudeAbsolute\argin,\textcol double precision\textgreater} luminosity\argin,\textcolor{red}{\textless integer\textgreater} field\argin)` Returns the maximum distance (in Mpc) at which a galaxy of the specified `mass` (in $M_\odot$) could be detected.

`double precision distanceMinimum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} magnitudeAbsolute\argin,\textcol double precision\textgreater} luminosity\argin,\textcolor{red}{\textless integer\textgreater} field\argin)` Returns the minimum distance (in Mpc) at which a galaxy of the specified `mass` (in $M_\odot$) would be included in the survey.

`integer fieldCount()` Returns the number of distinct fields included in the survey.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`logical pointIncluded(\textcolor{red}{\textless double precision\textgreater} point\argin,\textcolor{re double precision\textgreater} mass\argin)` Return true if the given Cartesian point lies within the survey bounds for the given mass limit.

`<void> randomsInitialize()` Initialize arrays of random points to define the survey angular geometry.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`double precision solidAngle(\textcolor{red}{\textless integer\textgreater} field\argin)` Return the solid angle (in steradians) of the survey.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision volumeMaximum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless integer\textgreater} field\argin)` Returns the maximum volume (in Mpc^3) at which a galaxy of the specified `mass` (in $M_\odot$) could be detected.

`logical windowFunctionAvailable()` Returns true if survey 3-D window functions are available.

`void windowFunctions(\textcolor{red}{\textless double precision\textgreater} mass1\argin,\textcolor{red}{\textless double precision\textgreater} mass2\argin,\textcolor{red}{\textless integer\textgreater} gridCount\argin,\textcolor{red}{\textless double precision\textgreater} boxLength\argout,\textcolor{red}{\textless complex((c_double_complex))\textgreater} windowFunction1\argout,\textcolor{red}{\textless complex((c_double_complex))\textgreater} windowFunction2\argout)` Returns the window functions on a grid of the specified size (`gridCount` cells in each dimension) for galaxies of the specified `mass1` and `mass2` (in $M_\odot$). The `boxLength` should be set to an appropriate value to fully enclose (with sufficient buffering to allow for Fourier transformation) the two window functions.

surveyGeometryMonteroDorta2009SDSS

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`double precision angularPower(\textcolor{red}{\textless integer\textgreater} i\argin,\textcolor{red}{\textless integer\textgreater} j\argin,\textcolor{red}{\textless integer\textgreater} l\argin)` Return C_ℓ^{ij}, where $(2\ell+1)C_\ell^{ij} = \sum_{m=-\ell}^{+\ell} \Psi_{\ell m}^i \Psi_{\ell m}^{j*}$, and $\Psi_{\ell m}^i$ are the cofficients of the spherical harmonic expansion of the i^{th} field.

`logical angularPowerAvailable()` Returns true if angular power spectrum of survey window function is available.

`integer angularPowerMaximumDegree()` Return the maximum degree, $\ell_{\max}$, for which the angular power is available.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((surveyGeometryClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\textcolor{red}{\textless logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision distanceMaximum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} magnitudeAbsolute\argin,\textcolor{red}{\textless double precision\textgreater} luminosity\argin,\textcolor{red}{\textless integer\textgreater} field\argin)` Returns the maximum distance (in Mpc) at which a galaxy of the specified `mass` (in $M_\odot$) could be detected.

`double precision distanceMinimum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} magnitudeAbsolute\argin,\textcolor{red}{\textless double precision\textgreater} luminosity\argin,\textcolor{red}{\textless integer\textgreater} field\argin)` Returns the minimum distance (in Mpc) at which a galaxy of the specified `mass` (in $M_\odot$) would be included in the survey.

`integer fieldCount()` Returns the number of distinct fields included in the survey.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges`
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<void> initialize()` Initialize an instance of the MANGLE survey geometry class.

`<logical> isDefault()` Return true if this is the default object of this class.

`<type(varying_string) > mangleDirectory()` Return the directory containing MANGLE files for this survey geometry.

`<type(varying_string)(:) > mangleFiles()` Return array of MANGLE filenames for this survey geometry.

`type(varying_string) objectType()` Return the type of the object.

`logical pointIncluded(\textcolor{red}{\textless double precision\textgreater} point\argin,\textcolor{re`
`double precision\textgreater} mass\argin)` Return true if the given Cartesian point lies within the survey bounds for the given mass limit.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`double precision solidAngle(\textcolor{red}{\textless integer\textgreater} field\argin)`
Return the solid angle (in steradians) of the survey.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl`
`type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-`
`size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles`
`type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-`
`size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision volumeMaximum(\textcolor{red}{\textless double precision\textgreater}`
`mass\argin,\textcolor{red}{\textless integer\textgreater} field\argin)` Returns the maximum volume (in Mpc^3) at which a galaxy of the specified `mass` (in $M_\odot$) could be detected.

`logical windowFunctionAvailable()` Returns true if survey 3-D window functions are available.

`void windowFunctions(\textcolor{red}{\textless double precision\textgreater} mass1\argin,\textcolor{red`
`double precision\textgreater} mass2\argin,\textcolor{red}{\textless integer\textgreater}`
`gridCount\argin,\textcolor{red}{\textless double precision\textgreater} boxLength\argout,\textcolor`
`complex((c_double_complex))\textgreater} windowFunction1\argout,\textcolor{red}{\textless`
`complex((c_double_complex))\textgreater} windowFunction2\argout)` Returns the window functions on a grid of the specified size (`gridCount` cells in each dimension) for galaxies of the specified `mass1` and `mass2` (in $M_\odot$). The `boxLength` should be set to an appropriate value to fully enclose (with sufficient buffering to allow for Fourier transformation) the two window functions.

surveyGeometryMoustakas2013PRIMUS

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

double precision angularPower(\textcolor{red}{\textless integer\textgreater} i\argin,\textcolor{red}{\t
integer\textgreater} j\argin,\textcolor{red}{\textless integer\textgreater} l\argin)
Return C_ℓ^{ij}, where $(2\ell+1)C_\ell^{ij} = \sum_{m=-\ell}^{+\ell} \Psi_{\ell m}^i \Psi_{\ell m}^{j*}$, and $\Psi_{\ell m}^i$ are the cofficients of the spherical harmonic expansion of the $i^{\rm th}$ field.

logical angularPowerAvailable() Returns true if angular power spectrum of survey window function is available.

integer angularPowerMaximumDegree() Return the maximum degree, $\ell_{\rm max}$, for which the angular power is available.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((surveyGeometryClass))\textgreater} destination\arginout)
Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision distanceMaximum(\textcolor{red}{\textless double precision\textgreater}
mass\argin,\textcolor{red}{\textless double precision\textgreater} magnitudeAbsolute\argin,\textcol
double precision\textgreater} luminosity\argin,\textcolor{red}{\textless integer\textgreater}
field\argin) Returns the maximum distance (in Mpc) at which a galaxy of the specified mass (in $M_\odot$) could be detected.

double precision distanceMinimum(\textcolor{red}{\textless double precision\textgreater}
mass\argin,\textcolor{red}{\textless double precision\textgreater} magnitudeAbsolute\argin,\textcol
double precision\textgreater} luminosity\argin,\textcolor{red}{\textless integer\textgreater}
field\argin) Returns the minimum distance (in Mpc) at which a galaxy of the specified mass (in $M_\odot$) would be included in the survey.

integer fieldCount() Returns the number of distinct fields included in the survey.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<void> initialize() Initialize an instance of the MANGLE survey geometry class.

<logical> isDefault() Return true if this is the default object of this class.

<type(varying_string) > mangleDirectory() Return the directory containing MANGLE files for this survey geometry.

<type(varying_string)(:) > mangleFiles() Return array of MANGLE filenames for this survey geometry.

type(varying_string) objectType() Return the type of the object.

`logical pointIncluded(\textcolor{red}{\textless double precision\textgreater} point\argin,\textcolor{re double precision\textgreater} mass\argin)` Return true if the given Cartesian point lies within the survey bounds for the given mass limit.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`double precision solidAngle(\textcolor{red}{\textless integer\textgreater} field\argin)` Return the solid angle (in steradians) of the survey.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision volumeMaximum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless integer\textgreater} field\argin)` Returns the maximum volume (in Mpc^3) at which a galaxy of the specified `mass` (in $M_\odot$) could be detected.

`logical windowFunctionAvailable()` Returns true if survey 3-D window functions are available.

`void windowFunctions(\textcolor{red}{\textless double precision\textgreater} mass1\argin,\textcolor{red double precision\textgreater} mass2\argin,\textcolor{red}{\textless integer\textgreater} gridCount\argin,\textcolor{red}{\textless double precision\textgreater} boxLength\argout,\textcolor complex((c_double_complex))\textgreater} windowFunction1\argout,\textcolor{red}{\textless complex((c_double_complex))\textgreater} windowFunction2\argout)` Returns the window functions on a grid of the specified size (`gridCount` cells in each dimension) for galaxies of the specified `mass1` and `mass2` (in $M_\odot$). The `boxLength` should be set to an appropriate value to fully enclose (with sufficient buffering to allow for Fourier transformation) the two window functions.

surveyGeometryMuzzin2013ULTRAVISTA

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`double precision angularPower(\textcolor{red}{\textless integer\textgreater} i\argin,\textcolor{red}{\t integer\textgreater} j\argin,\textcolor{red}{\textless integer\textgreater} l\argin)` Return C_ℓ^{ij}, where $(2\ell+1)C_\ell^{ij} = \sum_{m=-\ell}^{+\ell} \Psi_{\ell m}^i \Psi_{\ell m}^{j*}$, and $\Psi_{\ell m}^i$ are the cofficients of the spherical harmonic expansion of the i^{th} field.

`logical angularPowerAvailable()` Returns true if angular power spectrum of survey window function is available.

`integer angularPowerMaximumDegree()` Return the maximum degree, ℓ_{max}, for which the angular power is available.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((surveyGeometryClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision distanceMaximum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} magnitudeAbsolute\argin,\textcol double precision\textgreater} luminosity\argin,\textcolor{red}{\textless integer\textgreater} field\argin)` Returns the maximum distance (in Mpc) at which a galaxy of the specified `mass` (in $M_\odot$) could be detected.

`double precision distanceMinimum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} magnitudeAbsolute\argin,\textcol double precision\textgreater} luminosity\argin,\textcolor{red}{\textless integer\textgreater} field\argin)` Returns the minimum distance (in Mpc) at which a galaxy of the specified `mass` (in $M_\odot$) would be included in the survey.

`integer fieldCount()` Returns the number of distinct fields included in the survey.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<void> initialize()` Initialize an instance of the MANGLE survey geometry class.

`<logical> isDefault()` Return true if this is the default object of this class.

`<type(varying_string) > mangleDirectory()` Return the directory containing MANGLE files for this survey geometry.

`<type(varying_string)(:) > mangleFiles()` Return array of MANGLE filenames for this survey geometry.

`type(varying_string) objectType()` Return the type of the object.

`logical pointIncluded(\textcolor{red}{\textless double precision\textgreater} point\argin,\textcolor{re double precision\textgreater} mass\argin)` Return true if the given Cartesian point lies within the survey bounds for the given mass limit.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`double precision solidAngle(\textcolor{red}{\textless integer\textgreater} field\argin)` Return the solid angle (in steradians) of the survey.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision volumeMaximum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless integer\textgreater} field\argin)` Returns the maximum volume (in Mpc^3) at which a galaxy of the specified `mass` (in $M_\odot$) could be detected.

`logical windowFunctionAvailable()` Returns true if survey 3-D window functions are available.

`void windowFunctions(\textcolor{red}{\textless double precision\textgreater} mass1\argin,\textcolor{red double precision\textgreater} mass2\argin,\textcolor{red}{\textless integer\textgreater} gridCount\argin,\textcolor{red}{\textless double precision\textgreater} boxLength\argout,\textcolor complex((c_double_complex))\textgreater} windowFunction1\argout,\textcolor{red}{\textless complex((c_double_complex))\textgreater} windowFunction2\argout)` Returns the window functions on a grid of the specified size (`gridCount` cells in each dimension) for galaxies of the specified `mass1` and `mass2` (in $M_\odot$). The `boxLength` should be set to an appropriate value to fully enclose (with sufficient buffering to allow for Fourier transformation) the two window functions.

surveyGeometryRandomPoints

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`double precision angularPower(\textcolor{red}{\textless integer\textgreater} i\argin,\textcolor{red}{\t integer\textgreater} j\argin,\textcolor{red}{\textless integer\textgreater} l\argin)` Return C_ℓ^{ij}, where $(2\ell+1)C_\ell^{ij} = \sum_{m=-\ell}^{+\ell} \Psi_{\ell m}^i \Psi_{\ell m}^{j*}$, and $\Psi_{\ell m}^i$ are the cofficients of the spherical harmonic expansion of the i^{th} field.

`logical angularPowerAvailable()` Returns true if angular power spectrum of survey window function is available.

`integer angularPowerMaximumDegree()` Return the maximum degree, ℓ_{max}, for which the angular power is available.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((surveyGeometryClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision distanceMaximum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} magnitudeAbsolute\argin,\textcol double precision\textgreater} luminosity\argin,\textcolor{red}{\textless integer\textgreater} field\argin)` Returns the maximum distance (in Mpc) at which a galaxy of the specified `mass` (in $M_\odot$) could be detected.

`double precision distanceMinimum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} magnitudeAbsolute\argin,\textcol double precision\textgreater} luminosity\argin,\textcolor{red}{\textless integer\textgreater}`

field\argin) Returns the minimum distance (in Mpc) at which a galaxy of the specified mass (in $M_\odot$) would be included in the survey.

integer fieldCount() Returns the number of distinct fields included in the survey.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

logical pointIncluded(\textcolor{red}{\textless double precision\textgreater} point\argin,\textcolor{re double precision\textgreater} mass\argin) Return true if the given Cartesian point lies within the survey bounds for the given mass limit.

<void> randomsInitialize() Initialize arrays of random points to define the survey angular geometry.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

double precision solidAngle(\textcolor{red}{\textless integer\textgreater} field\argin) Return the solid angle (in steradians) of the survey.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision volumeMaximum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless integer\textgreater} field\argin) Returns the maximum volume (in Mpc^3) at which a galaxy of the specified mass (in $M_\odot$) could be detected.

logical windowFunctionAvailable() Returns true if survey 3-D window functions are available.

void windowFunctions(\textcolor{red}{\textless double precision\textgreater} mass1\argin,\textcolor{red double precision\textgreater} mass2\argin,\textcolor{red}{\textless integer\textgreater} gridCount\argin,\textcolor{red}{\textless double precision\textgreater} boxLength\argout,\textcolor complex((c_double_complex))\textgreater} windowFunction1\argout,\textcolor{red}{\textless complex((c_double_complex))\textgreater} windowFunction2\argout) Returns the window functions on a grid of the specified size (gridCount cells in each dimension) for galaxies of the specified mass1 and mass2 (in $M_\odot$). The boxLength should be set to an appropriate value to fully enclose (with sufficient buffering to allow for Fourier transformation) the two window functions.

surveyGeometryTomczak2014ZFOURGE

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`double precision angularPower(\textcolor{red}{\textless integer\textgreater} i\argin,\textcolor{red}{\t integer\textgreater} j\argin,\textcolor{red}{\textless integer\textgreater} l\argin)` Return C_ℓ^{ij}, where $(2\ell+1)C_\ell^{ij} = \sum_{m=-\ell}^{+\ell} \Psi_{\ell m}^i \Psi_{\ell m}^{j*}$, and $\Psi_{\ell m}^i$ are the cofficients of the spherical harmonic expansion of the $i^{\rm th}$ field.

`logical angularPowerAvailable()` Returns true if angular power spectrum of survey window function is available.

`integer angularPowerMaximumDegree()` Return the maximum degree, $\ell_{\rm max}$, for which the angular power is available.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((surveyGeometryClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision distanceMaximum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} magnitudeAbsolute\argin,\textcol double precision\textgreater} luminosity\argin,\textcolor{red}{\textless integer\textgreater} field\argin)` Returns the maximum distance (in Mpc) at which a galaxy of the specified `mass` (in $M_\odot$) could be detected.

`double precision distanceMinimum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} magnitudeAbsolute\argin,\textcol double precision\textgreater} luminosity\argin,\textcolor{red}{\textless integer\textgreater} field\argin)` Returns the minimum distance (in Mpc) at which a galaxy of the specified `mass` (in $M_\odot$) would be included in the survey.

`integer fieldCount()` Returns the number of distinct fields included in the survey.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<void> initialize()` Initialize an instance of the MANGLE survey geometry class.

`<logical> isDefault()` Return true if this is the default object of this class.

`<type(varying_string) > mangleDirectory()` Return the directory containing MANGLE files for this survey geometry.

`<type(varying_string)(:) > mangleFiles()` Return array of MANGLE filenames for this survey geometry.

`type(varying_string) objectType()` Return the type of the object.

logical pointIncluded(\textcolor{red}{\textless double precision\textgreater} point\argin,\textcolor{re
double precision\textgreater} mass\argin) Return true if the given Cartesian point lies within the survey bounds for the given mass limit.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

double precision solidAngle(\textcolor{red}{\textless integer\textgreater} field\argin) Return the solid angle (in steradians) of the survey.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision volumeMaximum(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless integer\textgreater} field\argin) Returns the maximum volume (in Mpc^3) at which a galaxy of the specified mass (in $M_\odot$) could be detected.

logical windowFunctionAvailable() Returns true if survey 3-D window functions are available.

void windowFunctions(\textcolor{red}{\textless double precision\textgreater} mass1\argin,\textcolor{red
double precision\textgreater} mass2\argin,\textcolor{red}{\textless integer\textgreater} gridCount\argin,\textcolor{red}{\textless double precision\textgreater} boxLength\argout,\textcolor
complex((c_double_complex))\textgreater} windowFunction1\argout,\textcolor{red}{\textless complex((c_double_complex))\textgreater} windowFunction2\argout) Returns the window functions on a grid of the specified size (gridCount cells in each dimension) for galaxies of the specified mass1 and mass2 (in $M_\odot$). The boxLength should be set to an appropriate value to fully enclose (with sufficient buffering to allow for Fourier transformation) the two window functions.

table

void deepCopyActions() Perform actions needed for deep copy of this object.

<void> destroy() Destroy the table.

void stateRestore(<integer> stateFile→,<type((fgsl_file))> fgslStateFile→,<integer> stateOperationID-
Restore the state of this object from file.

void stateStore(<integer> stateFile→,<type((fgsl_file))> fgslStateFile→,<integer> stateOperationID→)
Store the state of this object to file.

table1D

void deepCopyActions() Perform actions needed for deep copy of this object.

<void> destroy() Destroy the table.

`<double(:)> integrationWeights(<double> x0→, <double> x1→)` Return the weights to be applied to the table to integrate (using the trapezium rule) between `x0` and `x1`.

`<double> interpolate(<double> x,<integer> [table])` Interpolate to `x` in the `table`$^{\text{th}}$ table.

`<double> interpolateGradient(<double> x,<integer> [table])` Interpolate the gradient to `x` in the `table`$^{\text{th}}$ table.

`<logical> isMonotonic(<enumeration> [directionDecreasing|directionIncreasing],<logical> [allowEqual],<integer> [table])` Return true if the table y-values are monotonic. Optionally, the direction of monotonicity can be specified via the `direction` argument—by default either direction is allowed. By default consecutive equal values are considered non-monotonic. This behavior can be changed via the optional `allowEqual` argument. If `table` is specified then the `table`$^{\text{th}}$ table is used for the y-values, otherwise the first table is used.

`<void> reverse(<type(table)> reversedSelf,<integer> [table], <logical> [precise])` Reverse the table (i.e. swap x and y components) and return in `reversedSelf`. If `table` is specified then the `table`$^{\text{th}}$ table is used for the y-values, otherwise the first table is used. If the optional `precise` argument is set to `true` then the reversal must be precisely invertible—if this is not possible the method will abort.

`<integer> size()` Return the size (i.e. number of x-values) in the table.

`void stateRestore(<integer> stateFile→,<type((fgsl_file))> fgslStateFile→,<integer> stateOperationID-`
Restore the state of this object from file.

`void stateStore(<integer> stateFile→,<type((fgsl_file))> fgslStateFile→,<integer> stateOperationID→)`
Store the state of this object to file.

`<double> x(<integer> i)` Return the `i`$^{\text{th}}$ x-value.

`<double> xEffective(<double> x)` Return the effective value of x to use in table interpolations.

`<double(:)> xs(<integer> i)` Return an array of all x-values.

`<double> y(<integer> i,<integer> [table])` Return the `i`$^{\text{th}}$ y-value. If `table` is specified then the `table`$^{\text{th}}$ table is used for the y-values, otherwise the first table is used.

`<double(:)> ys(<integer> i,<integer> [table])` Return an array of all y-values. If `table` is specified then the `table`$^{\text{th}}$ table is used for the y-values, otherwise the first table is used.

table1DGeneric

`<void> create(<double(:)> x,<integer> [tableCount])` Create the object with the specified `x` values, and with `tableCount` tables.

`void deepCopyActions()` Perform actions needed for deep copy of this object.

`<void> destroy()` Destroy the table.

`<double(:)> integrationWeights(<double> x0→, <double> x1→)` Return the weights to be applied to the table to integrate (using the trapezium rule) between `x0` and `x1`.

`<double> interpolate(<double> x,<integer> [table])` Interpolate to `x` in the `table`$^{\text{th}}$ table.

`<double> interpolateGradient(<double> x,<integer> [table])` Interpolate the gradient to `x` in the `table`$^{\text{th}}$ table.

`<logical> isMonotonic(<enumeration> [directionDecreasing|directionIncreasing],<logical> [allowEqual],<integer> [table])` Return true if the table y-values are monotonic. Optionally, the direction of monotonicity can be specified via the `direction` argument—by default either direction is allowed. By default consecutive equal values are considered non-monotonic. This behavior can be changed via the optional `allowEqual` argument. If `table` is specified then the `table`$^{\text{th}}$ table is used for the y-values, otherwise the first table is used.

`<void> populate(<double>|<double(:)> y,<integer> [i],<integer> [table])` Populate the `table`$^{\text{th}}$ table with elements `y`. If `y` is a scalar, then the index, `i`, of the element to set must also be specified.

`<void> reverse(<type(table)> reversedSelf,<integer> [table], <logical> [precise])` Reverse the table (i.e. swap x and y components) and return in `reversedSelf`. If `table` is specified then the `table`$^{\text{th}}$ table is used for the y-values, otherwise the first table is used. If the optional `precise` argument is set to `true` then the reversal must be precisely invertible—if this is not possible the method will abort.

`<integer> size()` Return the size (i.e. number of x-values) in the table.

`void stateRestore(<integer> stateFile→,<type((fgsl_file))> fgslStateFile→,<integer> stateOperationID-`
Restore the state of this object from file.

`void stateStore(<integer> stateFile→,<type((fgsl_file))> fgslStateFile→,<integer> stateOperationID→)`
Store the state of this object to file.

`<double> x(<integer> i)` Return the `i`$^{\text{th}}$ x-value.

`<double> xEffective(<double> x)` Return the effective value of x to use in table interpolations.

`<double(:)> xs(<integer> i)` Return an array of all x-values.

`<double> y(<integer> i,<integer> [table])` Return the `i`$^{\text{th}}$ y-value. If `table` is specified then the `table`$^{\text{th}}$ table is used for the y-values, otherwise the first table is used.

`<double(:)> ys(<integer> i,<integer> [table])` Return an array of all y-values. If `table` is specified then the `table`$^{\text{th}}$ table is used for the y-values, otherwise the first table is used.

table1DLinearCSpline

`<void> create(<double> xMinimum,<double> xMaximum,<integer>xCount,<integer>[tableCount])`
Create the object with x-values spanning the range `xMinimum` to `xMaximum` in `xCount` steps, and with `tableCount` tables.

`void deepCopyActions()` Perform actions needed for deep copy of this object.

`<void> destroy()` Destroy the table.

`<double(:)> integrationWeights(<double> x0→, <double> x1→)` Return the weights to be applied to the table to integrate (using the trapezium rule) between `x0` and `x1`.

`<double> interpolate(<double> x,<integer> [table])` Interpolate to `x` in the `table`$^{\text{th}}$ table.

`<double> interpolateGradient(<double> x,<integer> [table])` Interpolate the gradient to `x` in the `table`$^{\text{th}}$ table.

`<logical> isMonotonic(<enumeration> [directionDecreasing|directionIncreasing],<logical> [allowEqual],<integer> [table])` Return true if the table y-values are monotonic. Optionally, the direction of monotonicity can be specified via the `direction` argument—by default either direction is allowed. By default consecutive equal values are considered non-monotonic. This behavior can be changed via the optional `allowEqual` argument. If `table` is specified then the `table`$^{\text{th}}$ table is used for the y-values, otherwise the first table is used.

`<void> populate(<double>|<double(:)> y,<integer> [i],<integer> [table])` Populate the `table`$^{\text{th}}$ table with elements `y`. If `y` is a scalar, then the index, `i`, of the element to set must also be specified.

`<void> reverse(<type(table)> reversedSelf,<integer> [table], <logical> [precise])` Reverse the table (i.e. swap x and y components) and return in `reversedSelf`. If `table` is specified then the `table`$^{\text{th}}$ table is used for the y-values, otherwise the first table is used. If the optional `precise` argument is set to `true` then the reversal must be precisely invertible—if this is not possible the method will abort.

`<integer> size()` Return the size (i.e. number of x-values) in the table.

`void stateRestore(<integer> stateFile→,<type((fgsl_file))> fgslStateFile→,<integer> stateOperationID-` Restore the state of this object from file.

`void stateStore(<integer> stateFile→,<type((fgsl_file))> fgslStateFile→,<integer> stateOperationID→)` Store the state of this object to file.

`<double> x(<integer> i)` Return the `i`$^{\text{th}}$ x-value.

`<double> xEffective(<double> x)` Return the effective value of x to use in table interpolations.

`<double(:)> xs(<integer> i)` Return an array of all x-values.

`<double> y(<integer> i,<integer> [table])` Return the `i`$^{\text{th}}$ y-value. If `table` is specified then the `table`$^{\text{th}}$ table is used for the y-values, otherwise the first table is used.

`<double(:)> ys(<integer> i,<integer> [table])` Return an array of all y-values. If `table` is specified then the `table`$^{\text{th}}$ table is used for the y-values, otherwise the first table is used.

table1DLinearLinear

`<void> create(<double> xMinimum,<double> xMaximum,<integer>xCount,<integer>[tableCount])` Create the object with x-values spanning the range `xMinimum` to `xMaximum` in `xCount` steps, and with `tableCount` tables.

`void deepCopyActions()` Perform actions needed for deep copy of this object.

`<void> destroy()` Destroy the table.

`<double(:)> integrationWeights(<double> x0→, <double> x1→)` Return the weights to be applied to the table to integrate (using the trapezium rule) between `x0` and `x1`.

`<double> interpolate(<double> x,<integer> [table])` Interpolate to `x` in the `table`$^{\text{th}}$ table.

`<double> interpolateGradient(<double> x,<integer> [table])` Interpolate the gradient to `x` in the `table`$^{\text{th}}$ table.

`<logical> isMonotonic(<enumeration> [directionDecreasing|directionIncreasing],<logical> [allowEqual],<integer> [table])` Return true if the table y-values are monotonic. Optionally, the direction of monotonicity can be specified via the `direction` argument—by default either direction is allowed. By default consecutive equal values are considered non-monotonic. This behavior can be changed via the optional `allowEqual` argument. If `table` is specified then the `table`$^{\text{th}}$ table is used for the y-values, otherwise the first table is used.

`<void> populate(<double>|<double(:)> y,<integer> [i],<integer> [table])` Populate the `table`$^{\text{th}}$ table with elements `y`. If `y` is a scalar, then the index, `i`, of the element to set must also be specified.

`<void> reverse(<type(table)> reversedSelf,<integer> [table], <logical> [precise])` Reverse the table (i.e. swap x and y components) and return in `reversedSelf`. If `table` is specified then the `table`$^{\text{th}}$ table is used for the y-values, otherwise the first table is used. If the optional `precise` argument is set to `true` then the reversal must be precisely invertible—if this is not possible the method will abort.

`<integer> size()` Return the size (i.e. number of x-values) in the table.

`void stateRestore(<integer> stateFile→,<type((fgsl_file))> fgslStateFile→,<integer> stateOperationID-` Restore the state of this object from file.

`void stateStore(<integer> stateFile→,<type((fgsl_file))> fgslStateFile→,<integer> stateOperationID→)` Store the state of this object to file.

`<double> x(<integer> i)` Return the `i`$^{\text{th}}$ x-value.

`<double> xEffective(<double> x)` Return the effective value of x to use in table interpolations.

`<double(:)> xs(<integer> i)` Return an array of all x-values.

`<double> y(<integer> i,<integer> [table])` Return the `i`$^{\text{th}}$ y-value. If `table` is specified then the `table`$^{\text{th}}$ table is used for the y-values, otherwise the first table is used.

`<double(:)> ys(<integer> i,<integer> [table])` Return an array of all y-values. If `table` is specified then the `table`$^{\text{th}}$ table is used for the y-values, otherwise the first table is used.

table1DLinearMonotoneCSpline

`<void> create(<double> xMinimum,<double> xMaximum,<integer>xCount,<integer>[tableCount])` Create the object with x-values spanning the range `xMinimum` to `xMaximum` in `xCount` steps, and with `tableCount` tables.

`void deepCopyActions()` Perform actions needed for deep copy of this object.

`<void> destroy()` Destroy the table.

`<double(:)> integrationWeights(<double> x0→, <double> x1→)` Return the weights to be applied to the table to integrate (using the trapezium rule) between `x0` and `x1`.

`<double> interpolate(<double> x,<integer> [table])` Interpolate to `x` in the `table`$^{\text{th}}$ table.

`<double> interpolateGradient(<double> x,<integer> [table])` Interpolate the gradient to `x` in the `table`$^{\text{th}}$ table.

`<logical> isMonotonic(<enumeration> [directionDecreasing|directionIncreasing],<logical> [allowEqual],<integer> [table])` Return true if the table y-values are monotonic. Optionally, the direction of monotonicity can be specified via the `direction` argument—by default either direction is allowed. By default consecutive equal values are considered non-monotonic. This behavior can be changed via the optional `allowEqual` argument. If `table` is specified then the `table`$^{\text{th}}$ table is used for the y-values, otherwise the first table is used.

`<void> populate(<double>|<double(:)> y,<integer> [i],<integer> [table])` Populate the `table`$^{\text{th}}$ table with elements `y`. If `y` is a scalar, then the index, `i`, of the element to set must also be specified.

`<void> reverse(<type(table)> reversedSelf,<integer> [table], <logical> [precise])` Reverse the table (i.e. swap x and y components) and return in `reversedSelf`. If `table` is specified then the `table`$^{\text{th}}$ table is used for the y-values, otherwise the first table is used. If the optional `precise` argument is set to `true` then the reversal must be precisely invertible—if this is not possible the method will abort.

`<integer> size()` Return the size (i.e. number of x-values) in the table.

`void stateRestore(<integer> stateFile→,<type((fgsl_file))> fgslStateFile→,<integer> stateOperationID-` Restore the state of this object from file.

`void stateStore(<integer> stateFile→,<type((fgsl_file))> fgslStateFile→,<integer> stateOperationID→)` Store the state of this object to file.

`<double> x(<integer> i)` Return the `i`$^{\text{th}}$ x-value.

`<double> xEffective(<double> x)` Return the effective value of x to use in table interpolations.

`<double(:)> xs(<integer> i)` Return an array of all x-values.

`<double> y(<integer> i,<integer> [table])` Return the `i`$^{\text{th}}$ y-value. If `table` is specified then the `table`$^{\text{th}}$ table is used for the y-values, otherwise the first table is used.

`<double(:)> ys(<integer> i,<integer> [table])` Return an array of all y-values. If `table` is specified then the `table`$^{\text{th}}$ table is used for the y-values, otherwise the first table is used.

table1DLogarithmicCSpline

`<void> create(<double> xMinimum,<double> xMaximum,<integer>xCount,<integer>[tableCount])` Create the object with x-values spanning the range `xMinimum` to `xMaximum` in `xCount` steps, and with `tableCount` tables.

`void deepCopyActions()` Perform actions needed for deep copy of this object.

`<void> destroy()` Destroy the table.

`<double(:)> integrationWeights(<double> x0→, <double> x1→)` Return the weights to be applied to the table to integrate (using the trapezium rule) between `x0` and `x1`.

`<double> interpolate(<double> x,<integer> [table])` Interpolate to `x` in the `table`$^{\text{th}}$ table.

`<double> interpolateGradient(<double> x,<integer> [table])` Interpolate the gradient to `x` in the `table`$^{\text{th}}$ table.

<logical> isMonotonic(<enumeration> [directionDecreasing|directionIncreasing],<logical> [allowEqual],<integer> [table]) Return true if the table y-values are monotonic. Optionally, the direction of monotonicity can be specified via the direction argument—by default either direction is allowed. By default consecutive equal values are considered non-monotonic. This behavior can be changed via the optional allowEqual argument. If table is specified then the tableth table is used for the y-values, otherwise the first table is used.

<void> populate(<double>|<double(:)> y,<integer> [i],<integer> [table]) Populate the tableth table with elements y. If y is a scalar, then the index, i, of the element to set must also be specified.

<void> reverse(<type(table)> reversedSelf,<integer> [table], <logical> [precise]) Reverse the table (i.e. swap x and y components) and return in reversedSelf. If table is specified then the tableth table is used for the y-values, otherwise the first table is used. If the optional precise argument is set to true then the reversal must be precisely invertible—if this is not possible the method will abort.

<integer> size() Return the size (i.e. number of x-values) in the table.

void stateRestore(<integer> stateFile→,<type((fgsl_file))> fgslStateFile→,<integer> stateOperationID- Restore the state of this object from file.

void stateStore(<integer> stateFile→,<type((fgsl_file))> fgslStateFile→,<integer> stateOperationID→) Store the state of this object to file.

<double> x(<integer> i) Return the ith x-value.

<double> xEffective(<double> x) Return the effective value of x to use in table interpolations.

<double(:)> xs(<integer> i) Return an array of all x-values.

<double> y(<integer> i,<integer> [table]) Return the ith y-value. If table is specified then the tableth table is used for the y-values, otherwise the first table is used.

<double(:)> ys(<integer> i,<integer> [table]) Return an array of all y-values. If table is specified then the tableth table is used for the y-values, otherwise the first table is used.

table1DLogarithmicLinear

<void> create(<double> xMinimum,<double> xMaximum,<integer>xCount,<integer>[tableCount]) Create the object with x-values spanning the range xMinimum to xMaximum in xCount steps, and with tableCount tables.

void deepCopyActions() Perform actions needed for deep copy of this object.

<void> destroy() Destroy the table.

<double(:)> integrationWeights(<double> x0→, <double> x1→) Return the weights to be applied to the table to integrate (using the trapezium rule) between x0 and x1.

<double> interpolate(<double> x,<integer> [table]) Interpolate to x in the tableth table.

<double> interpolateGradient(<double> x,<integer> [table]) Interpolate the gradient to x in the tableth table.

`<logical> isMonotonic(<enumeration> [directionDecreasing|directionIncreasing],<logical> [allowEqual],<integer> [table])` Return true if the table y-values are monotonic. Optionally, the direction of monotonicity can be specified via the `direction` argument—by default either direction is allowed. By default consecutive equal values are considered non-monotonic. This behavior can be changed via the optional `allowEqual` argument. If `table` is specified then the `table`$^{\text{th}}$ table is used for the y-values, otherwise the first table is used.

`<void> populate(<double>|<double(:)> y,<integer> [i],<integer> [table])` Populate the `table`$^{\text{th}}$ table with elements `y`. If `y` is a scalar, then the index, `i`, of the element to set must also be specified.

`<void> reverse(<type(table)> reversedSelf,<integer> [table], <logical> [precise])` Reverse the table (i.e. swap x and y components) and return in `reversedSelf`. If `table` is specified then the `table`$^{\text{th}}$ table is used for the y-values, otherwise the first table is used. If the optional `precise` argument is set to `true` then the reversal must be precisely invertible—if this is not possible the method will abort.

`<integer> size()` Return the size (i.e. number of x-values) in the table.

`void stateRestore(<integer> stateFile→,<type((fgsl_file))> fgslStateFile→,<integer> stateOperationID-` Restore the state of this object from file.

`void stateStore(<integer> stateFile→,<type((fgsl_file))> fgslStateFile→,<integer> stateOperationID→)` Store the state of this object to file.

`<double> x(<integer> i)` Return the `i`$^{\text{th}}$ x-value.

`<double> xEffective(<double> x)` Return the effective value of x to use in table interpolations.

`<double(:)> xs(<integer> i)` Return an array of all x-values.

`<double> y(<integer> i,<integer> [table])` Return the `i`$^{\text{th}}$ y-value. If `table` is specified then the `table`$^{\text{th}}$ table is used for the y-values, otherwise the first table is used.

`<double(:)> ys(<integer> i,<integer> [table])` Return an array of all y-values. If `table` is specified then the `table`$^{\text{th}}$ table is used for the y-values, otherwise the first table is used.

table1DLogarithmicMonotoneCSpline

`<void> create(<double> xMinimum,<double> xMaximum,<integer>xCount,<integer>[tableCount])` Create the object with x-values spanning the range `xMinimum` to `xMaximum` in `xCount` steps, and with `tableCount` tables.

`void deepCopyActions()` Perform actions needed for deep copy of this object.

`<void> destroy()` Destroy the table.

`<double(:)> integrationWeights(<double> x0→, <double> x1→)` Return the weights to be applied to the table to integrate (using the trapezium rule) between `x0` and `x1`.

`<double> interpolate(<double> x,<integer> [table])` Interpolate to `x` in the `table`$^{\text{th}}$ table.

`<double> interpolateGradient(<double> x,<integer> [table])` Interpolate the gradient to `x` in the `table`$^{\text{th}}$ table.

<logical> isMonotonic(<enumeration> [directionDecreasing|directionIncreasing],<logical> [allowEqual],<integer> [table]) Return true if the table y-values are monotonic. Optionally, the direction of monotonicity can be specified via the direction argument—by default either direction is allowed. By default consecutive equal values are considered non-monotonic. This behavior can be changed via the optional allowEqual argument. If table is specified then the tableth table is used for the y-values, otherwise the first table is used.

<void> populate(<double>|<double(:)> y,<integer> [i],<integer> [table]) Populate the tableth table with elements y. If y is a scalar, then the index, i, of the element to set must also be specified.

<void> reverse(<type(table)> reversedSelf,<integer> [table], <logical> [precise]) Reverse the table (i.e. swap x and y components) and return in reversedSelf. If table is specified then the tableth table is used for the y-values, otherwise the first table is used. If the optional precise argument is set to true then the reversal must be precisely invertible—if this is not possible the method will abort.

<integer> size() Return the size (i.e. number of x-values) in the table.

void stateRestore(<integer> stateFile→,<type((fgsl_file))> fgslStateFile→,<integer> stateOperationID- Restore the state of this object from file.

void stateStore(<integer> stateFile→,<type((fgsl_file))> fgslStateFile→,<integer> stateOperationID→) Store the state of this object to file.

<double> x(<integer> i) Return the ith x-value.

<double> xEffective(<double> x) Return the effective value of x to use in table interpolations.

<double(:)> xs(<integer> i) Return an array of all x-values.

<double> y(<integer> i,<integer> [table]) Return the ith y-value. If table is specified then the tableth table is used for the y-values, otherwise the first table is used.

<double(:)> ys(<integer> i,<integer> [table]) Return an array of all y-values. If table is specified then the tableth table is used for the y-values, otherwise the first table is used.

table1DNonUniformLinearLogarithmic

<void> create(<double(:)> x,<integer> [tableCount]) Create the object with the specified x values, and with tableCount tables.

void deepCopyActions() Perform actions needed for deep copy of this object.

<void> destroy() Destroy the table.

<double(:)> integrationWeights(<double> x0→, <double> x1→) Return the weights to be applied to the table to integrate (using the trapezium rule) between x0 and x1.

<double> interpolate(<double> x,<integer> [table]) Interpolate to x in the tableth table.

<double> interpolateGradient(<double> x,<integer> [table]) Interpolate the gradient to x in the tableth table.

`<logical> isMonotonic(<enumeration> [directionDecreasing|directionIncreasing],<logical> [allowEqual],<integer> [table])` Return true if the table y-values are monotonic. Optionally, the direction of monotonicity can be specified via the `direction` argument—by default either direction is allowed. By default consecutive equal values are considered non-monotonic. This behavior can be changed via the optional `allowEqual` argument. If `table` is specified then the `table`$^{\text{th}}$ table is used for the y-values, otherwise the first table is used.

`<void> populate(<double>|<double(:)> y,<integer> [i],<integer> [table])` Populate the `table`$^{\text{th}}$ table with elements `y`. If `y` is a scalar, then the index, `i`, of the element to set must also be specified.

`<void> reverse(<type(table)> reversedSelf,<integer> [table], <logical> [precise])` Reverse the table (i.e. swap x and y components) and return in `reversedSelf`. If `table` is specified then the `table`$^{\text{th}}$ table is used for the y-values, otherwise the first table is used. If the optional `precise` argument is set to `true` then the reversal must be precisely invertible—if this is not possible the method will abort.

`<integer> size()` Return the size (i.e. number of x-values) in the table.

`void stateRestore(<integer> stateFile→,<type((fgsl_file))> fgslStateFile→,<integer> stateOperationID-` Restore the state of this object from file.

`void stateStore(<integer> stateFile→,<type((fgsl_file))> fgslStateFile→,<integer> stateOperationID→)` Store the state of this object to file.

`<double> x(<integer> i)` Return the `i`$^{\text{th}}$ x-value.

`<double> xEffective(<double> x)` Return the effective value of x to use in table interpolations.

`<double(:)> xs(<integer> i)` Return an array of all x-values.

`<double> y(<integer> i,<integer> [table])` Return the `i`$^{\text{th}}$ y-value. If `table` is specified then the `table`$^{\text{th}}$ table is used for the y-values, otherwise the first table is used.

`<double(:)> ys(<integer> i,<integer> [table])` Return an array of all y-values. If `table` is specified then the `table`$^{\text{th}}$ table is used for the y-values, otherwise the first table is used.

table2DLinLinLin

`<void> create(<double(:)> x,<double(:)> y,<integer> [tableCount])` Create the object with the specified `x` and `y` values, and with `tableCount` tables.

`void deepCopyActions()` Perform actions needed for deep copy of this object.

`<void> destroy()` Destroy the table.

`<double> interpolate(<double> x,<double> y,<integer> [table])` Interpolate to `x`, `y` in the `table`$^{\text{th}}$ table.

`<void> populate(<double>|<double(:,:)> z,<integer> [i],<integer> [j],<integer> [table])` Populate the `table`$^{\text{th}}$ table with elements `y`. If `y` is a scalar, then the indices, `i`, `j`, of the element to set must also be specified.

`void stateRestore(<integer> stateFile→,<type((fgsl_file))> fgslStateFile→,<integer> stateOperationID-` Restore the state of this object from file.

`void stateStore(<integer> stateFile→,<type((fgsl_file))> fgslStateFile→,<integer> stateOperationID→)` Store the state of this object to file.

`<double(:)> xs()` Return an array of all `x` values.

`<double(:)> ys()` Return an array of all `y` values.

`<double(:,:,:)> zs()` Return an array of all `z` values.

table2DLogLogLin

`<void> create(<double> xMinimum,<double> xMaximum,<double> yMinimum,<double> yMaximum,<integer>yCount,<` Create the object with x-values spanning the range `xMinimum` to `xMaximum` in `xCount` steps, and with `tableCount` tables.

`void deepCopyActions()` Perform actions needed for deep copy of this object.

`<void> destroy()` Destroy the table.

`<double> interpolate(<double> x,<integer> [table])` Interpolate to `x` in the `table`$^{\text{th}}$ table.

`<double> interpolateGradient(<double> x,<integer> [table])` Interpolate the gradient to `x` in the `table`$^{\text{th}}$ table.

`<void> interpolationFactors(<double> x, <double> y)` Compute and store interpolation factors to `(x,y)`.

`<logical> isInitialized()` Return true if the table is initialized (this means the table is created, it may not yet have been populated).

`<void> populate(<double>|<double(:,:)> z,<integer> [i],<integer> [j],<integer> [table])` Populate the `table`$^{\text{th}}$ table with elements `y`. If `y` is a scalar, then the index, `i`, of the element to set must also be specified.

`<integer> size(<integer> dim)` Return the size (i.e. number of x or y-values) in the table of the given dimension.

`void stateRestore(<integer> stateFile→,<type((fgsl_file))> fgslStateFile→,<integer> stateOperationID-` Restore the state of this object from file.

`void stateStore(<integer> stateFile→,<type((fgsl_file))> fgslStateFile→,<integer> stateOperationID→)` Store the state of this object to file.

`<double> x(<integer> i)` Return the `i`$^{\text{th}}$ x-value.

`<double(:)> xs()` Return an array of all x-values.

`<double> y(<integer> i,<integer> [table])` Return the `i`$^{\text{th}}$ y-value. If `table` is specified then the `table`$^{\text{th}}$ table is used for the y-values, otherwise the first table is used.

`<double(:)> ys()` Return an array of all y-values. If `table` is specified then the `table`$^{\text{th}}$ table is used for the y-values, otherwise the first table is used.

`<double> z(<integer> i,<integer> j,<integer> [table])` Return the `(i,j)`$^{\text{th}}$ z-value. If `table` is specified then the `table`$^{\text{th}}$ table is used for the z-values, otherwise the first table is used.

`<double(:,:)> zs(<integer> [table])` Return an array of all z-values. If `table` is specified then the `table`$^{\text{th}}$ table is used for the z-values, otherwise the first table is used.

taskAGNSpectraHopkins2008BuildFile

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((taskClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

void perform(\textcolor{red}{\textless integer\textgreater} status\argout) Perform the task.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

logical requiresOutputFile() Should return true if the task requires the main output file to be open.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

taskBuildToolCAMB

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((taskClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

void perform(\textcolor{red}{\textless integer\textgreater} status\argout) Perform the task.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

logical requiresOutputFile() Should return true if the task requires the main output file to be open.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

taskBuildToolCloudy

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((taskClass))\textgreater} destination\arginout)
Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

void perform(\textcolor{red}{\textless integer\textgreater} status\argout) Perform the task.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

`logical requiresOutputFile()` Should return true if the task requires the main output file to be open.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

taskBuildToolFSPS

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((taskClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`void perform(\textcolor{red}{\textless integer\textgreater} status\argout)` Perform the task.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`logical requiresOutputFile()` Should return true if the task requires the main output file to be open.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

taskBuildToolRecFast

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((taskClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`void perform(\textcolor{red}{\textless integer\textgreater} status\argout)` Perform the task.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`logical requiresOutputFile()` Should return true if the task requires the main output file to be open.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

taskCatalogProjectedCorrelationFunction

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((taskClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

void perform(\textcolor{red}{\textless integer\textgreater} status\argout) Perform the task.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

logical requiresOutputFile() Should return true if the task requires the main output file to be open.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

taskClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((taskClass))\textgreater} destination\arginout)
Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

void perform(\textcolor{red}{\textless integer\textgreater} status\argout) Perform the task.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

`logical requiresOutputFile()` Should return true if the task requires the main output file to be open.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

taskConditionalMassFunction

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((taskClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`void perform(\textcolor{red}{\textless integer\textgreater} status\argout)` Perform the task.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`logical requiresOutputFile()` Should return true if the task requires the main output file to be open.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

taskEvolveForests

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((taskClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

void perform(\textcolor{red}{\textless integer\textgreater} status\argout) Perform the task.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

logical requiresOutputFile() Should return true if the task requires the main output file to be open.

<void> resumeTree(<type(mergerTree)> tree←) Restore a suspended tree.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

<void> suspendTree(<type(mergerTree)> tree←) Suspend a tree (to memory or to file).

taskExcursionSets

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((taskClass))\textgreater} destination\arginout) Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`void perform(\textcolor{red}{\textless integer\textgreater} status\argout)` Perform the task.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`logical requiresOutputFile()` Should return true if the task requires the main output file to be open.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

taskHaloMassFunction

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((taskClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`void perform(\textcolor{red}{\textless integer\textgreater} status\argout)` Perform the task.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

logical requiresOutputFile() Should return true if the task requires the main output file to be open.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

taskHaloModelGenerate

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((taskClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

void perform(\textcolor{red}{\textless integer\textgreater} status\argout) Perform the task.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

logical requiresOutputFile() Should return true if the task requires the main output file to be open.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

taskHaloModelProjectedCorrelationFunction

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((taskClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`void perform(\textcolor{red}{\textless integer\textgreater} status\argout)` Perform the task.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`logical requiresOutputFile()` Should return true if the task requires the main output file to be open.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

taskHaloSpinDistribution

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((taskClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

void perform(\textcolor{red}{\textless integer\textgreater} status\argout) Perform the task.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

logical requiresOutputFile() Should return true if the task requires the main output file to be open.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

taskIntergalacticMediumState

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((taskClass))\textgreater} destination\arginout)
Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

void perform(\textcolor{red}{\textless integer\textgreater} status\argout) Perform the task.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

logical requiresOutputFile() Should return true if the task requires the main output file to be open.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

taskLocalGroupDatabase

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((taskClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

void perform(\textcolor{red}{\textless integer\textgreater} status\argout) Perform the task.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

logical requiresOutputFile() Should return true if the task requires the main output file to be open.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

taskMassFunctionCovariance

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((taskClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

void perform(\textcolor{red}{\textless integer\textgreater} status\argout) Perform the task.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

logical requiresOutputFile() Should return true if the task requires the main output file to be open.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

taskMergerTreeFileBuilder

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((taskClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

void perform(\textcolor{red}{\textless integer\textgreater} status\argout) Perform the task.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

logical requiresOutputFile() Should return true if the task requires the main output file to be open.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

taskMulti

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((taskClass))\textgreater} destination\arginout)
Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

void perform(\textcolor{red}{\textless integer\textgreater} status\argout) Perform the task.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

logical requiresOutputFile() Should return true if the task requires the main output file to be open.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

taskNBodyAnalyze

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((taskClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

void perform(\textcolor{red}{\textless integer\textgreater} status\argout) Perform the task.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

logical requiresOutputFile() Should return true if the task requires the main output file to be open.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

taskPosteriorSample

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((taskClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`void perform(\textcolor{red}{\textless integer\textgreater} status\argout)` Perform the task.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`logical requiresOutputFile()` Should return true if the task requires the main output file to be open.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

taskPowerSpectra

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((taskClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

void perform(\textcolor{red}{\textless integer\textgreater} status\argout) Perform the task.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

logical requiresOutputFile() Should return true if the task requires the main output file to be open.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

taskReport

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((taskClass))\textgreater} destination\arginout)
Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

void perform(\textcolor{red}{\textless integer\textgreater} status\argout) Perform the task.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

`logical requiresOutputFile()` Should return true if the task requires the main output file to be open.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

tensorRank2Dimension3Symmetric

`<type(tensorRank2Dimension3Symmetric)> add(<type(tensorRank2Dimension3Symmetric)> tensorRank2Dimension3` Add two tensors.

`<void> builder(<*type(node)> tensorRank2Dimension3SymmetricDefinition→)` Build tensor object from a provided XML description.

`<double> contract()` Contract a tensor, returning $\mathbf{T}_i^i$.

`<void> deserialize(<double(:)> tensorArray→)` Deserialize the tensor object from an array.

`<void> destroy()` Destroy a tensor object.

`<type(tensorRank2Dimension3Symmetric)> divide(<double> divisor→)` Divide a tensor by a scalar.

`<double> doubleContract(<type(tensorRank2Dimension3Symmetric)> tensor1)` Return the double contraction of two tensors, $\mathbf{A}_j^i\mathbf{B}_i^j$.

`<void> dump()` Dump the tensor object.

`<void> dumpRaw(<integer> fileHandle→)` Dump the tensor object to binary.

`<logical> equality(<type(tensorRank2Dimension3Symmetric)> self→, <double(3,3)> matrix→)` Return true if a tensor and a matrix are equal.

`<type(tensorRank2Dimension3Symmetric)> fromMatrix(double(3,3) matrix)` Construct a matrix from a tensor object.

`<void> increment(<type(tensorRank2Dimension3Symmetric)> addTensor→)` Increment the tensor object.

`<logical> isZero()` Return true if a tensor object is zero.

`<type(tensorRank2Dimension3Symmetric)> multiply(<double> multiplier→)` Multiply a tensor by a scalar.

`<integer(c_size_t) > nonStaticSizeOf()` Returns the size of any non-static components of the type.

`<void> readRaw(<integer> fileHandle→)` Read the tensor object from binary.

`<void> reset()` Reset elements to zero.

`<void> serialize(<double(:)> tensorArray←)` Serialize the tensor object to an array.

`<integer> serializeCount()` Return a count of the number of properties in a serialized tensor object.

<void> setToIdentity() Set a tensor object to the indentity (i.e. all diagonal elements 1, all other elements 0).

<void> setToUnity() Set all elements of the tensor object to unity.

<type(tensorRank2Dimension3Symmetric)> subtract(<type(tensorRank2Dimension3Symmetric)> tensor1→, <type(tensorRank2Dimension3Symmetric)> tensor2→) Subtract one tensor from another.

double(3,3) toMatrix() Construct a matrix from a tensor object.

tidalStrippingDisksClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((tidalStrippingDisksClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision rateMassLoss(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the rate of mass loss (in $M_\odot$ Gyr^{-1}) due to tidal stripping of the disk component of node.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

tidalStrippingDisksNull

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((tidalStrippingDisksClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`double precision rateMassLoss(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)` Returns the rate of mass loss (in $M_\odot$ Gyr^{-1}) due to tidal stripping of the disk component of `node`.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

tidalStrippingDisksSimple

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((tidalStrippingDisksClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision rateMassLoss(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)
Returns the rate of mass loss (in $M_\odot$ Gyr^{-1}) due to tidal stripping of the disk component of node.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

tidalStrippingSpheroidsClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((tidalStrippingSpheroidsClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision rateMassLoss(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout)
Returns the rate of mass loss (in $M_\odot$ Gyr^{-1}) due to tidal stripping of the spheroid component of node.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

tidalStrippingSpheroidsNull

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((tidalStrippingSpheroidsClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision rateMassLoss(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the rate of mass loss (in $M_\odot$ Gyr^{-1}) due to tidal stripping of the spheroid component of node.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

tidalStrippingSpheroidsSimple

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((tidalStrippingSpheroidsClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

double precision rateMassLoss(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout) Returns the rate of mass loss (in $M_\odot$ Gyr^{-1}) due to tidal stripping of the spheroid component of node.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

transferFunctionAccelerator

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((transferFunctionClass))\textgreater} destination\arginou
Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

`double precision epochTime()` Return the cosmic time corresponding to the epoch for which this transfer function is defined.

`double precision halfModeMass(\textcolor{red}{\textless integer\textgreater} status\argout)` Return the mass (in $M_\odot$) corresponding to the wavenumber at which the transfer function is suppressed by a factor of two due to small-scale dark matter particle physics.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision logarithmicDerivative(\textcolor{red}{\textless double precision\textgreater} wavenumber\argin)` Return the logarithmic derivative of the transfer function for $k =$`wavenumber` [Mpc^{-1}].

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision value(\textcolor{red}{\textless double precision\textgreater} wavenumber\argin)` Return the transfer function for $k =$`wavenumber` [Mpc^{-1}].

transferFunctionBBKS

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((transferFunctionClass))\textgreater} destination\arginou` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision epochTime()` Return the cosmic time corresponding to the epoch for which this transfer function is defined.

`double precision halfModeMass(\textcolor{red}{\textless integer\textgreater} status\argout)` Return the mass (in $M_\odot$) corresponding to the wavenumber at which the transfer function is suppressed by a factor of two due to small-scale dark matter particle physics.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`double precision logarithmicDerivative(\textcolor{red}{\textless double precision\textgreater} wavenumber\argin)` Return the logarithmic derivative of the transfer function for $k =$`wavenumber` [Mpc^{-1}].

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision value(\textcolor{red}{\textless double precision\textgreater} wavenumber\argin)` Return the transfer function for $k =$`wavenumber` [Mpc^{-1}].

transferFunctionBBKSWDM

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((transferFunctionClass))\textgreater} destination\arginou` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision epochTime()` Return the cosmic time corresponding to the epoch for which this transfer function is defined.

`double precision halfModeMass(\textcolor{red}{\textless integer\textgreater} status\argout)` Return the mass (in $M_\odot$) corresponding to the wavenumber at which the transfer function is suppressed by a factor of two due to small-scale dark matter particle physics.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision logarithmicDerivative(\textcolor{red}{\textless double precision\textgreater} wavenumber\argin) Return the logarithmic derivative of the transfer function for $k =$wavenumber [Mpc^{-1}].

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision value(\textcolor{red}{\textless double precision\textgreater} wavenumber\argin) Return the transfer function for $k =$wavenumber [Mpc^{-1}].

transferFunctionBode2001

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((transferFunctionClass))\textgreater} destination\arginou Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision epochTime() Return the cosmic time corresponding to the epoch for which this transfer function is defined.

double precision halfModeMass(\textcolor{red}{\textless integer\textgreater} status\argout) Return the mass (in $M_\odot$) corresponding to the wavenumber at which the transfer function is suppressed by a factor of two due to small-scale dark matter particle physics.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

`double precision logarithmicDerivative(\textcolor{red}{\textless double precision\textgreater} wavenumber\argin)` Return the logarithmic derivative of the transfer function for $k =$`wavenumber` [Mpc^{-1}].

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision value(\textcolor{red}{\textless double precision\textgreater} wavenumber\argin)` Return the transfer function for $k =$`wavenumber` [Mpc^{-1}].

transferFunctionCAMB

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void checkRange(<double>wavenumber→)` Check that the provided wavenumber is within the tabulated range of the transfer function.

`void deepCopy(\textcolor{red}{\textless class((transferFunctionClass))\textgreater} destination\arginou` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`double precision epochTime()` Return the cosmic time corresponding to the epoch for which this transfer function is defined.

`double precision halfModeMass(\textcolor{red}{\textless integer\textgreater} status\argout)` Return the mass (in $M_\odot$) corresponding to the wavenumber at which the transfer function is suppressed by a factor of two due to small-scale dark matter particle physics.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

double precision logarithmicDerivative(\textcolor{red}{\textless double precision\textgreater} wavenumber\argin) Return the logarithmic derivative of the transfer function for $k =$wavenumber [Mpc^{-1}].

type(varying_string) objectType() Return the type of the object.

void readFile(<char(len=*)> fileName→) Read the named transfer function file.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision value(\textcolor{red}{\textless double precision\textgreater} wavenumber\argin) Return the transfer function for $k =$wavenumber [Mpc^{-1}].

transferFunctionClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((transferFunctionClass))\textgreater} destination\arginou Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision epochTime() Return the cosmic time corresponding to the epoch for which this transfer function is defined.

double precision halfModeMass(\textcolor{red}{\textless integer\textgreater} status\argout) Return the mass (in $M_\odot$) corresponding to the wavenumber at which the transfer function is suppressed by a factor of two due to small-scale dark matter particle physics.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision logarithmicDerivative(\textcolor{red}{\textless double precision\textgreater} wavenumber\argin) Return the logarithmic derivative of the transfer function for $k =$wavenumber [Mpc^{-1}].

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision value(\textcolor{red}{\textless double precision\textgreater} wavenumber\argin) Return the transfer function for $k =$wavenumber [Mpc^{-1}].

transferFunctionEisensteinHu1999

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

<void> computeFactors(<double> wavenumber→, <double> wavenumberEffective←, <double> wavenumberNeutrir <double> L←, <double> C←) Compute common factors needed by Eisenstein and Hu [1999] transfer function calculations.

void deepCopy(\textcolor{red}{\textless class((transferFunctionClass))\textgreater} destination\arginou Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision epochTime() Return the cosmic time corresponding to the epoch for which this transfer function is defined.

double precision halfModeMass(\textcolor{red}{\textless integer\textgreater} status\argout) Return the mass (in $M_\odot$) corresponding to the wavenumber at which the transfer function is suppressed by a factor of two due to small-scale dark matter particle physics.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision logarithmicDerivative(\textcolor{red}{\textless double precision\textgreater} wavenumber\argin) Return the logarithmic derivative of the transfer function for $k =$wavenumber [Mpc^{-1}].

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision value(\textcolor{red}{\textless double precision\textgreater} wavenumber\argin) Return the transfer function for $k =$wavenumber [Mpc^{-1}].

transferFunctionFile

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((transferFunctionClass))\textgreater} destination\arginou Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision epochTime() Return the cosmic time corresponding to the epoch for which this transfer function is defined.

double precision halfModeMass(\textcolor{red}{\textless integer\textgreater} status\argout) Return the mass (in $M_\odot$) corresponding to the wavenumber at which the transfer function is suppressed by a factor of two due to small-scale dark matter particle physics.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision logarithmicDerivative(\textcolor{red}{\textless double precision\textgreater} wavenumber\argin) Return the logarithmic derivative of the transfer function for $k =$wavenumber [Mpc^{-1}].

type(varying_string) objectType() Return the type of the object.

void readFile(<char(len=*)> fileName→) Read the named transfer function file.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision value(\textcolor{red}{\textless double precision\textgreater} wavenumber\argin) Return the transfer function for $k =$wavenumber [Mpc^{-1}].

transferFunctionIdentity

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((transferFunctionClass))\textgreater} destination\arginou Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision epochTime() Return the cosmic time corresponding to the epoch for which this transfer function is defined.

double precision halfModeMass(\textcolor{red}{\textless integer\textgreater} status\argout) Return the mass (in $M_\odot$) corresponding to the wavenumber at which the transfer function is suppressed by a factor of two due to small-scale dark matter particle physics.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

double precision logarithmicDerivative(\textcolor{red}{\textless double precision\textgreater} wavenumber\argin) Return the logarithmic derivative of the transfer function for $k =$wavenumber [Mpc^{-1}].

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision value(\textcolor{red}{\textless double precision\textgreater} wavenumber\argin)` Return the transfer function for $k =$`wavenumber` [Mpc^{-1}].

treeNode

`<class(nodeComponentAgeStatistics), pointer => component> ageStatistics(<integer> [instance]→, <logical> [autoCreate]→)` Return a `ageStatistics` component member of the node. If no `instance` is specified, return the first instance. If `autoCreate` is `true` then create a single instance of the component if none exists in the node.

`<integer> ageStatisticsCount()` Returns the number of `ageStatistics` components in the node.

`<void> ageStatisticsCreate(<class(nodeComponentAgeStatistics)> [template]→)` Create the `ageStatistics` component of `self`.

`<void> ageStatisticsDestroy()` Destroy the `ageStatistics` component of `self`

`<void> ageStatisticsMove(<type(treeNode)> targetNode↔, <logical> [overwrite]→)` Move instances of the `ageStatistics` component, from one node to another.

`<void> ageStatisticsRemove(<integer> instance→)` Remove an instance of the `ageStatistics` component from a node.

`<void> attachEvent(<*class(nodeEvent)> newEvent↔)` Attach a `nodeEvent` object to this node.

`<class(nodeComponentBasic), pointer => component> basic(<integer> [instance]→, <logical> [autoCreate]→)` Return a `basic` component member of the node. If no `instance` is specified, return the first instance. If `autoCreate` is `true` then create a single instance of the component if none exists in the node.

`<integer> basicCount()` Returns the number of `basic` components in the node.

`<void> basicCreate(<class(nodeComponentBasic)> [template]→)` Create the `basic` component of `self`.

`<void> basicDestroy()` Destroy the `basic` component of `self`

`<void> basicMove(<type(treeNode)> targetNode↔, <logical> [overwrite]→)` Move instances of the `basic` component, from one node to another.

`<void> basicRemove(<integer> instance→)` Remove an instance of the `basic` component from a node.

`<class(nodeComponentBlackHole), pointer => component> blackHole(<integer> [instance]→, <logical> [autoCreate]→)` Return a `blackHole` component member of the node. If no `instance` is specified, return the first instance. If `autoCreate` is `true` then create a single instance of the component if none exists in the node.

<integer> blackHoleCount() Returns the number of blackHole components in the node.

<void> blackHoleCreate(<class(nodeComponentBlackHole)> [template]→) Create the blackHole component of self.

<void> blackHoleDestroy() Destroy the blackHole component of self

<void> blackHoleMove(<type(treeNode)> targetNode↔, <logical> [overwrite]→) Move instances of the blackHole component, from one node to another.

<void> blackHoleRemove(<integer> instance→) Remove an instance of the blackHole component from a node.

<void> componentBuilder(<*type(node)> nodeDefinition→) Build components in a treeNode object given an XML definition.

<void> copyNodeTo(<class(treeNode)> targetNode↔, <logical> [skipFormationNode]→, <logical> [skipEvent]→) Make a copy of self in targetNode.

<class(nodeComponentDarkMatterProfile), pointer => component> darkMatterProfile(<integer> [instance]→, <logical> [autoCreate]→) Return a darkMatterProfile component member of the node. If no instance is specified, return the first instance. If autoCreate is true then create a single instance of the component if none exists in the node.

<integer> darkMatterProfileCount() Returns the number of darkMatterProfile components in the node.

<void> darkMatterProfileCreate(<class(nodeComponentDarkMatterProfile)> [template]→) Create the darkMatterProfile component of self.

<void> darkMatterProfileDestroy() Destroy the darkMatterProfile component of self

<void> darkMatterProfileMove(<type(treeNode)> targetNode↔, <logical> [overwrite]→) Move instances of the darkMatterProfile component, from one node to another.

<void> darkMatterProfileRemove(<integer> instance→) Remove an instance of the darkMatterProfile component from a node.

<void> deserializeRates(<double precision[:]> array→, <integer> propertyType→) Deserialize rates of evolvable properties of a node from an array.

<void> deserializeRaw(<integer> fileHandle→) Deserialize a tree node object from a raw (binary) file.

<void> deserializeValues(<double precision[:]> array→, <integer> propertyType→) Deserialize evolvable properties of a node from an array.

<void> destroy() Destroy a treeNode object.

<void> destroyBranch() Destroy a branch of a merger tree rooted at this node.

<class(nodeComponentDisk), pointer => component> disk(<integer> [instance]→, <logical> [autoCreate]→) Return a disk component member of the node. If no instance is specified, return the first instance. If autoCreate is true then create a single instance of the component if none exists in the node.

<integer> diskCount() Returns the number of disk components in the node.

<void> diskCreate(<class(nodeComponentDisk)> [template]→) Create the disk component of self.

<void> diskDestroy() Destroy the disk component of self

<void> diskMove(<type(treeNode)> targetNode↔, <logical> [overwrite]→) Move instances of the disk component, from one node to another.

<void> diskRemove(<integer> instance→) Remove an instance of the disk component from a node.

<class(nodeComponentDynamicsStatistics), pointer => component> dynamicsStatistics(<integer> [instance]→, <logical> [autoCreate]→) Return a dynamicsStatistics component member of the node. If no instance is specified, return the first instance. If autoCreate is true then create a single instance of the component if none exists in the node.

<integer> dynamicsStatisticsCount() Returns the number of dynamicsStatistics components in the node.

<void> dynamicsStatisticsCreate(<class(nodeComponentDynamicsStatistics)> [template]→) Create the dynamicsStatistics component of self.

<void> dynamicsStatisticsDestroy() Destroy the dynamicsStatistics component of self

<void> dynamicsStatisticsMove(<type(treeNode)> targetNode↔, <logical> [overwrite]→) Move instances of the dynamicsStatistics component, from one node to another.

<void> dynamicsStatisticsRemove(<integer> instance→) Remove an instance of the dynamicsStatistics component from a node.

<*type(treeNode)> earliestProgenitor() Return a pointer to the earliest progenitor (along the main branch) of this node.

<class(nodeComponentFormationTime), pointer => component> formationTime(<integer> [instance]→, <logical> [autoCreate]→) Return a formationTime component member of the node. If no instance is specified, return the first instance. If autoCreate is true then create a single instance of the component if none exists in the node.

<integer> formationTimeCount() Returns the number of formationTime components in the node.

<void> formationTimeCreate(<class(nodeComponentFormationTime)> [template]→) Create the formationTime component of self.

<void> formationTimeDestroy() Destroy the formationTime component of self

<void> formationTimeMove(<type(treeNode)> targetNode↔, <logical> [overwrite]→) Move instances of the formationTime component, from one node to another.

<void> formationTimeRemove(<integer> instance→) Remove an instance of the formationTime component from a node.

<class(nodeComponentHostHistory), pointer => component> hostHistory(<integer> [instance]→, <logical> [autoCreate]→) Return a hostHistory component member of the node. If no instance is specified, return the first instance. If autoCreate is true then create a single instance of the component if none exists in the node.

<integer> hostHistoryCount() Returns the number of hostHistory components in the node.

<void> hostHistoryCreate(<class(nodeComponentHostHistory)> [template]→) Create the hostHistory component of self.

<void> hostHistoryDestroy() Destroy the hostHistory component of self

<void> hostHistoryMove(<type(treeNode)> targetNode↔, <logical> [overwrite]→) Move instances of the hostHistory component, from one node to another.

<void> hostHistoryRemove(<integer> instance→) Remove an instance of the hostHistory component from a node.

<class(nodeComponentHotHalo), pointer => component> hotHalo(<integer> [instance]→, <logical> [autoCreate]→) Return a hotHalo component member of the node. If no instance is specified, return the first instance. If autoCreate is true then create a single instance of the component if none exists in the node.

<integer> hotHaloCount() Returns the number of hotHalo components in the node.

<void> hotHaloCreate(<class(nodeComponentHotHalo)> [template]→) Create the hotHalo component of self.

<void> hotHaloDestroy() Destroy the hotHalo component of self

<void> hotHaloMove(<type(treeNode)> targetNode↔, <logical> [overwrite]→) Move instances of the hotHalo component, from one node to another.

<void> hotHaloRemove(<integer> instance→) Remove an instance of the hotHalo component from a node.

<integer(kind_int8)> index() Return the index of this node.

<void> indexSet(<integer(kind_int8)> index→) Set the index of this node.

<class(nodeComponentIndices), pointer => component> indices(<integer> [instance]→, <logical> [autoCreate]→) Return a indices component member of the node. If no instance is specified, return the first instance. If autoCreate is true then create a single instance of the component if none exists in the node.

<integer> indicesCount() Returns the number of indices components in the node.

<void> indicesCreate(<class(nodeComponentIndices)> [template]→) Create the indices component of self.

<void> indicesDestroy() Destroy the indices component of self

<void> indicesMove(<type(treeNode)> targetNode↔, <logical> [overwrite]→) Move instances of the indices component, from one node to another.

<void> indicesRemove(<integer> instance→) Remove an instance of the indices component from a node.

<void> initialize(<integer(kind=kind_int8)> [index]→, <type(mergerTree)> [hostTree]→) Initialize a treeNode object.

<class(nodeComponentInterOutput), pointer => component> interOutput(<integer> [instance]→, <logical> [autoCreate]→) Return a interOutput component member of the node. If no instance is specified, return the first instance. If autoCreate is true then create a single instance of the component if none exists in the node.

<integer> interOutputCount() Returns the number of interOutput components in the node.

<void> interOutputCreate(<class(nodeComponentInterOutput)> [template]→) Create the interOutput component of self.

<void> interOutputDestroy() Destroy the interOutput component of self

<void> interOutputMove(<type(treeNode)> targetNode↔, <logical> [overwrite]→) Move instances of the interOutput component, from one node to another.

<void> interOutputRemove(<integer> instance→) Remove an instance of the interOutput component from a node.

<logical> isOnMainBranch() Return true if this node is on the main branch of its tree, false otherwise.

<logical> isPrimaryProgenitor() Return true if this node is the primary progenitor of its descendent, false otherwise.

<logical> isPrimaryProgenitorOf(<integer(kind_int8)> targetNodeIndex→|<*type(treeNode)> targetNode→) Return true is this node is the primary progenitor of the specified (by index or pointer) node, false otherwise.

<logical> isProgenitorOf(<integer(kind_int8)> targetNodeIndex→|<*type(treeNode)> targetNode→) Return true is this node is a progenitor of the specified (by index or pointer) node, false otherwise.

<logical> isSatellite() Return true if this node is a satellite, false otherwise.

<*type(treeNode)> lastSatellite() Return a pointer to the last satellite in the list of satellites beloning to this node.

<double precision> mapDouble0(<integer> reduction→, <integer> [optimizeFor]→) Map a rank-0, double function over components of the node.

<void> mapVoid() Map a void function over components of the node.

<class(nodeComponentMassFlowStatistics), pointer => component> massFlowStatistics(<integer> [instance]→, <logical> [autoCreate]→) Return a massFlowStatistics component member of the node. If no instance is specified, return the first instance. If autoCreate is true then create a single instance of the component if none exists in the node.

<integer> massFlowStatisticsCount() Returns the number of massFlowStatistics components in the node.

<void> massFlowStatisticsCreate(<class(nodeComponentMassFlowStatistics)> [template]→) Create the massFlowStatistics component of self.

<void> massFlowStatisticsDestroy() Destroy the massFlowStatistics component of self

<void> massFlowStatisticsMove(<type(treeNode)> targetNode↔, <logical> [overwrite]→) Move instances of the massFlowStatistics component, from one node to another.

<void> massFlowStatisticsRemove(<integer> instance→) Remove an instance of the massFlowStatistics component from a node.

<*type(treeNode)> mergesWith() Return a pointer to the node with which this node will merge.

<class(nodeComponentMergingStatistics), pointer => component> mergingStatistics(<integer> [instance]→, <logical> [autoCreate]→) Return a mergingStatistics component member of the node. If no instance is specified, return the first instance. If autoCreate is true then create a single instance of the component if none exists in the node.

<integer> mergingStatisticsCount() Returns the number of mergingStatistics components in the node.

<void> mergingStatisticsCreate(<class(nodeComponentMergingStatistics)> [template]→) Create the mergingStatistics component of self.

<void> mergingStatisticsDestroy() Destroy the mergingStatistics component of self

<void> mergingStatisticsMove(<type(treeNode)> targetNode↔, <logical> [overwrite]→) Move instances of the mergingStatistics component, from one node to another.

<void> mergingStatisticsRemove(<integer> instance→) Remove an instance of the mergingStatistics component from a node.

<void> moveComponentsTo(<type(treeNode)> targetNode↔) Move components from self to targetNode.

<type(varying_string)> nameFromIndex(<integer> index→, <integer> propertyType→) Return the name of a property given its index within an array.

<class(nodeComponentNBody), pointer => component> nBody(<integer> [instance]→, <logical> [autoCreate]→) Return a nBody component member of the node. If no instance is specified, return the first instance. If autoCreate is true then create a single instance of the component if none exists in the node.

<integer> nBodyCount() Returns the number of nBody components in the node.

<void> nBodyCreate(<class(nodeComponentNBody)> [template]→) Create the nBody component of self.

<void> nBodyDestroy() Destroy the nBody component of self

<void> nBodyMove(<type(treeNode)> targetNode↔, <logical> [overwrite]→) Move instances of the nBody component, from one node to another.

<void> nBodyRemove(<integer> instance→) Remove an instance of the nBody component from a node.

<void> odeStepInactivesInitialize() Initialize the inactives in components of tree node self in preparation for an ODE solver step.

<void> odeStepRatesInitialize() Initialize the rates in components of tree node self in preparation for an ODE solver step.

<void> odeStepScalesInitialize() Initialize the scales in components of tree node self in preparation for an ODE solver step.

<void> output(<integer> integerProperty↔, <integer> integerBufferCount↔, <integer(kind=kind_int8)[:,:]> integerBuffer↔, <integer> doubleProperty↔, <integer> doubleBufferCount↔, <double precision[:,:]> doubleBuffer↔, <double precision> time→, <type(multiCounter)> outputInstance→) Populate output buffers with properties to output for a treeNode.

<void> outputCount(<integer> integerPropertyCount↔, <integer> doublePropertyCount↔, <double precision> time→) Increment the count of properties to output for a treeNode.

<void> outputNames(<integer> integerProperty↔, <character(len=*)[:]> integerPropertyNames↔, <character(len=*)[:]> integerPropertyComments↔, <double precision[:]> integerPropertyUnitsSI↔, <integer> doubleProperty↔, <character(len=*)[:]> doublePropertyNames↔, <character(len=*)[:]> doublePropertyComments↔, <double precision[:]> doublePropertyUnitsSI↔, <double precision> time→) Establish the names of properties to output for a treeNode.

<class(nodeComponentPosition), pointer => component> position(<integer> [instance]→, <logical> [autoCreate]→) Return a position component member of the node. If no instance is specified, return the first instance. If autoCreate is true then create a single instance of the component if none exists in the node.

<integer> positionCount() Returns the number of position components in the node.

<void> positionCreate(<class(nodeComponentPosition)> [template]→) Create the position component of self.

<void> positionDestroy() Destroy the position component of self

<void> positionMove(<type(treeNode)> targetNode↔, <logical> [overwrite]→) Move instances of the position component, from one node to another.

<void> positionRemove(<integer> instance→) Remove an instance of the position component from a node.

<void> postOutput(<double precision> time→) Perform post-output processing of a treeNode.

<void> removeFromHost() Remove this node from the satellite population of its host halo.

<void> removeFromMergee() Remove this node from the list of mergees associated with its merge target.

<void> removePairedEvent(<class(nodeEvent)> event→) Remove a paired nodeEvent from this node.

<class(nodeComponentSatellite), pointer => component> satellite(<integer> [instance]→, <logical> [autoCreate]→) Return a satellite component member of the node. If no instance is specified, return the first instance. If autoCreate is true then create a single instance of the component if none exists in the node.

<integer> satelliteCount() Returns the number of satellite components in the node.

<void> satelliteCreate(<class(nodeComponentSatellite)> [template]→) Create the satellite component of self.

<void> satelliteDestroy() Destroy the satellite component of self

<void> satelliteMove(<type(treeNode)> targetNode↔, <logical> [overwrite]→) Move instances of the satellite component, from one node to another.

<void> satelliteRemove(<integer> instance→) Remove an instance of the satellite component from a node.

<void> serializationOffsets(<integer> propertyType→) Compute offsets into serialization arrays for treeNode object.

<void> serializeASCII() Serialize node content to ASCII.

<integer> serializeCount(<integer> propertyType→) Return a count of the size of the node when serialized to an array.

<void> serializeInactives(<logical[:]> array←) Serialize inactive statuses of evolvable properties of a node into an array.

<void> serializeRates(<double precision[:]> array←, <integer> propertyType→) Serialize rates of evolvable properties of a node into an array.

<void> serializeRaw(<integer> fileHandle→) Serialize all content of a tree node to a raw (binary) file.

<void> serializeScales(<double precision[:]> array←, <integer> propertyType→) Serialize scales of evolvable properties of a node into an array.

<void> serializeValues(<double precision[:]> array←, <integer> propertyType→) Serialize evolvable properties of a node into an array.

<void> serializeXML(<integer> fileHandle→) Serialize tree node content as XML.

<integer(c_size_t)> sizeOf() Compute the size (in bytes) of the tree node.

<class(nodeComponentSpheroid), pointer => component> spheroid(<integer> [instance]→, <logical> [autoCreate]→) Return a spheroid component member of the node. If no instance is specified, return the first instance. If autoCreate is true then create a single instance of the component if none exists in the node.

<integer> spheroidCount() Returns the number of spheroid components in the node.

<void> spheroidCreate(<class(nodeComponentSpheroid)> [template]→) Create the spheroid component of self.

<void> spheroidDestroy() Destroy the spheroid component of self

<void> spheroidMove(<type(treeNode)> targetNode↔, <logical> [overwrite]→) Move instances of the spheroid component, from one node to another.

<void> spheroidRemove(<integer> instance→) Remove an instance of the spheroid component from a node.

<class(nodeComponentSpin), pointer => component> spin(<integer> [instance]→, <logical> [autoCreate]→) Return a spin component member of the node. If no instance is specified, return the first instance. If autoCreate is true then create a single instance of the component if none exists in the node.

<integer> spinCount() Returns the number of spin components in the node.

<void> spinCreate(<class(nodeComponentSpin)> [template]→) Create the spin component of self.

<void> spinDestroy() Destroy the spin component of self

<void> spinMove(<type(treeNode)> targetNode↔, <logical> [overwrite]→) Move instances of the spin component, from one node to another.

<void> spinRemove(<integer> instance→) Remove an instance of the spin component from a node.

<double> timeStep() Return the time-step last used by this node.

<void> timeStepSet(<double>index→) Set the time-step used by this node.

<type(varying_string)> type() Return the type of this node.

<integer(kind_int8)> uniqueID() Return the unique identifier for this node.

<void> uniqueIDSet(<integer(kind_int8)> uniqueID→) Set the unique identifier for this node.

<void> walkBranchWithSatellites(<*type(treeNode)> startNode↔) Return a pointer to the next node when performing a walk of a single branch of the tree, including satellites.

<void> walkTreeWithSatellites() Return a pointer to the next node when performing a walk of the entire tree, including satellites.

unevolvedSubhaloMassFunctionClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((unevolvedSubhaloMassFunctionClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision differential(\textcolor{red}{\textless double precision\textgreater} time\argin,\texto double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} massHost\argin) Return the differential unevolved subhalo mass function per halo for mass $[M_\odot]$ subhalos in massHost $[M_\odot]$ hosts at time [Gyr].

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

double precision integrated(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcol double precision\textgreater} massLow\argin,\textcolor{red}{\textless double precision\textgreater} massHigh\argin,\textcolor{red}{\textless double precision\textgreater} massHost\argin) Return the unevolved subhalo mass function per host at time [Gyr] in hosts of mass massHost $[M_\odot]$ integrated between massLow and massHigh $[M_\odot]$.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

unevolvedSubhaloMassFunctionGiocoli2008

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((unevolvedSubhaloMassFunctionClass))\textgreater} destination\arginout) Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

double precision differential(\textcolor{red}{\textless double precision\textgreater} time\argin,\textc double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} massHost\argin) Return the differential unevolved subhalo mass function per halo for mass $[M_\odot]$ subhalos in massHost $[M_\odot]$ hosts at time [Gyr].

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

double precision integrated(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcol double precision\textgreater} massLow\argin,\textcolor{red}{\textless double precision\textgreater} massHigh\argin,\textcolor{red}{\textless double precision\textgreater} massHost\argin) Return the unevolved subhalo mass function per host at time [Gyr] in hosts of mass massHost $[M_\odot]$ integrated between massLow and massHigh $[M_\odot]$.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

universe

`<*type(universeEvent)> createEvent()` Create a `treeEvent` object in this universe.

`<*type(mergerTree)> popTree()` Pop a `mergerTree` from this universe.

`<void> pushTree(<*type(mergerTree)> thisTree→)` Pop a `mergerTree` from this universe.

`<void> removeEvent(<type(universeEvent)> event→)` Remove a `treeEvent` from this universe.

universeOperatorClass

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((universeOperatorClass))\textgreater} destination\arginou` Perform a deep copy of the object.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`type(varying_string) objectType()` Return the type of the object.

`void operate(\textcolor{red}{\textless type((universe))\textgreater} universe_\arginout)` Operate on the universe.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

universeOperatorIdentity

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((universeOperatorClass))\textgreater} destination\arginou Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

void operate(\textcolor{red}{\textless type((universe))\textgreater} universe_\arginout) Operate on the universe.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

universeOperatorIntergalacticMediumStateEvolve

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((universeOperatorClass))\textgreater} destination\arginou Perform a deep copy of the object.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

void operate(\textcolor{red}{\textless type((universe))\textgreater} universe_\arginout)
Operate on the universe.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateSet(<integer(c_size_t) > iNow→) Set the state of the IGM state class up to the given time index.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

vector

<type(vector)> add(<class(vector)> vector1→, <class(vector)> vector2→) Compute vector1+vector2.

<type(vector)> subtract(<class(vector)> vector1→, <class(vector)> vector2→) Compute vector1-vector2.

virialDensityContrastBryanNorman1998

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((virialDensityContrastClass))\textgreater}
destination\arginout) Perform a deep copy of the object.

double precision densityContrast(\textcolor{red}{\textless double precision\textgreater}
mass\argin,\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\text
double precision\textgreater} expansionFactor\argin,\textcolor{red}{\textless logical\textgreater}
collapsing\argin) Returns the virial density contrast at the given epoch.

double precision densityContrastRateOfChange(\textcolor{red}{\textless double precision\textgreater}
mass\argin,\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\text
double precision\textgreater} expansionFactor\argin,\textcolor{red}{\textless logical\textgreater}
collapsing\argin) Returns the rate of change of virial density contrast at the given epoch.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`logical isMassDependent()` Returns true if the virial density contrast is mass-dependent.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision turnAroundOverVirialRadii(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin,\textcolor logical\textgreater} collapsing\argin)` Returns the ratio of turnaround and virial radii at the given epoch.

virialDensityContrastClass

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((virialDensityContrastClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`double precision densityContrast(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\text double precision\textgreater} expansionFactor\argin,\textcolor{red}{\textless logical\textgreater} collapsing\argin)` Returns the virial density contrast at the given epoch.

`double precision densityContrastRateOfChange(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\text double precision\textgreater} expansionFactor\argin,\textcolor{red}{\textless logical\textgreater} collapsing\argin)` Returns the rate of change of virial density contrast at the given epoch.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`logical isMassDependent()` Returns true if the virial density contrast is mass-dependent.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision turnAroundOverVirialRadii(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin,\textcolor logical\textgreater} collapsing\argin)` Returns the ratio of turnaround and virial radii at the given epoch.

virialDensityContrastFixed

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((virialDensityContrastClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`double precision densityContrast(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\text double precision\textgreater} expansionFactor\argin,\textcolor{red}{\textless logical\textgreater} collapsing\argin)` Returns the virial density contrast at the given epoch.

`double precision densityContrastRateOfChange(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\text double precision\textgreater} expansionFactor\argin,\textcolor{red}{\textless logical\textgreater} collapsing\argin)` Returns the rate of change of virial density contrast at the given epoch.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`logical isMassDependent()` Returns true if the virial density contrast is mass-dependent.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision turnAroundOverVirialRadii(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin,\textcolor logical\textgreater} collapsing\argin)` Returns the ratio of turnaround and virial radii at the given epoch.

virialDensityContrastFriendsOfFriends

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((virialDensityContrastClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`double precision densityContrast(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\text double precision\textgreater} expansionFactor\argin,\textcolor{red}{\textless logical\textgreater} collapsing\argin)` Returns the virial density contrast at the given epoch.

`double precision densityContrastRateOfChange(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\text double precision\textgreater} expansionFactor\argin,\textcolor{red}{\textless logical\textgreater} collapsing\argin)` Returns the rate of change of virial density contrast at the given epoch.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`logical isMassDependent()` Returns true if the virial density contrast is mass-dependent.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision turnAroundOverVirialRadii(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin,\textcolor logical\textgreater} collapsing\argin)` Returns the ratio of turnaround and virial radii at the given epoch.

virialDensityContrastKitayamaSuto1996

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((virialDensityContrastClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`double precision densityContrast(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\text double precision\textgreater} expansionFactor\argin,\textcolor{red}{\textless logical\textgreater} collapsing\argin)` Returns the virial density contrast at the given epoch.

`double precision densityContrastRateOfChange(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\text double precision\textgreater} expansionFactor\argin,\textcolor{red}{\textless logical\textgreater} collapsing\argin)` Returns the rate of change of virial density contrast at the given epoch.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`logical isMassDependent()` Returns true if the virial density contrast is mass-dependent.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision turnAroundOverVirialRadii(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin,\textcolor logical\textgreater} collapsing\argin)` Returns the ratio of turnaround and virial radii at the given epoch.

virialDensityContrastPercolation

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((virialDensityContrastClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`double precision densityContrast(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\text double precision\textgreater} expansionFactor\argin,\textcolor{red}{\textless logical\textgreater} collapsing\argin)` Returns the virial density contrast at the given epoch.

`double precision densityContrastRateOfChange(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\text double precision\textgreater} expansionFactor\argin,\textcolor{red}{\textless logical\textgreater} collapsing\argin)` Returns the rate of change of virial density contrast at the given epoch.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`logical isMassDependent()` Returns true if the virial density contrast is mass-dependent.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`<void> tabulate(<double> mass→, <double> time→)` Tabulate the virial density contrast as a function of mass and time.

`double precision turnAroundOverVirialRadii(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin,\textcolor logical\textgreater} collapsing\argin)` Returns the ratio of turnaround and virial radii at the given epoch.

virialDensityContrastSphericalCollapseMatterDE

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((virialDensityContrastClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`double precision densityContrast(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\text double precision\textgreater} expansionFactor\argin,\textcolor{red}{\textless logical\textgreater} collapsing\argin)` Returns the virial density contrast at the given epoch.

`double precision densityContrastRateOfChange(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\text double precision\textgreater} expansionFactor\argin,\textcolor{red}{\textless logical\textgreater} collapsing\argin)` Returns the rate of change of virial density contrast at the given epoch.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`logical isMassDependent()` Returns true if the virial density contrast is mass-dependent.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void retabulate(<double> time→)` Tabulate spherical collapse virial density contrast.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision turnAroundOverVirialRadii(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin,\textcolor logical\textgreater} collapsing\argin)` Returns the ratio of turnaround and virial radii at the given epoch.

virialDensityContrastSphericalCollapseMatterLambda

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((virialDensityContrastClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`double precision densityContrast(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\text double precision\textgreater} expansionFactor\argin,\textcolor{red}{\textless logical\textgreater} collapsing\argin)` Returns the virial density contrast at the given epoch.

`double precision densityContrastRateOfChange(\textcolor{red}{\textless double precision\textgreater} mass\argin,\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\text double precision\textgreater} expansionFactor\argin,\textcolor{red}{\textless logical\textgreater} collapsing\argin)` Returns the rate of change of virial density contrast at the given epoch.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

`<logical> isDefault()` Return true if this is the default object of this class.

`logical isMassDependent()` Returns true if the virial density contrast is mass-dependent.

`type(varying_string) objectType()` Return the type of the object.

`<integer> referenceCountDecrement()` Decrement the reference count to this object and return the new reference count.

`<void> referenceCountIncrement()` Increment the reference count to this object.

`<void> referenceCountReset()` Reset the reference count to this object to 0.

`void retabulate(<double> time→)` Tabulate spherical collapse virial density contrast.

`void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Restore the state of this object from file.

`void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin)` Store the state of this object to file.

`double precision turnAroundOverVirialRadii(\textcolor{red}{\textless double precision\textgreater} time\argin,\textcolor{red}{\textless double precision\textgreater} expansionFactor\argin,\textcolor logical\textgreater} collapsing\argin)` Returns the ratio of turnaround and virial radii at the given epoch.

virialOrbitBenson2005

`void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin)` Return a list of parameter names allowed for this object.

`void autoHook()` Insert any event hooks required by this object.

`void deepCopy(\textcolor{red}{\textless class((virialOrbitClass))\textgreater} destination\arginout)` Perform a deep copy of the object.

`class(virialDensityContrastClass) densityContrastDefinition()` Returns a `virialDensityContrast` object describing the virial density contrast used to define this orbit class.

`void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin)` Return an input parameter list descriptor which could be used to recreate this object.

`type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges` Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

type(keplerOrbit) orbit(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolo type((treeNode))\textgreater} host\arginout,\textcolor{red}{\textless logical\textgreater} acceptUnboundOrbits\argin) Returns an orbit object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision velocityTangentialMagnitudeMean(\textcolor{red}{\textless type((treeNode))\textgreater node\arginout,\textcolor{red}{\textless type((treeNode))\textgreater} host\arginout) Returns the mean of the magnitude of tangential velocity averaged over all orbits.

double precision, dimension(3) velocityTangentialVectorMean(\textcolor{red}{\textless type((treeNode))\ node\arginout,\textcolor{red}{\textless type((treeNode))\textgreater} host\arginout) Returns the mean vector of the vector tangential velocity averaged over all orbits.

double precision velocityTotalRootMeanSquared(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless type((treeNode))\textgreater} host\arginout) Returns the square root of the mean of the squared total velocity averaged over all orbits.

virialOrbitClass

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((virialOrbitClass))\textgreater} destination\arginout) Perform a deep copy of the object.

class(virialDensityContrastClass) densityContrastDefinition() Returns a virialDensityContrast object describing the virial density contrast used to define this orbit class.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

type(keplerOrbit) orbit(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolo type((treeNode))\textgreater} host\arginout,\textcolor{red}{\textless logical\textgreater} acceptUnboundOrbits\argin) Returns an orbit object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision velocityTangentialMagnitudeMean(\textcolor{red}{\textless type((treeNode))\textgreater node\arginout,\textcolor{red}{\textless type((treeNode))\textgreater} host\arginout) Returns the mean of the magnitude of tangential velocity averaged over all orbits.

double precision, dimension(3) velocityTangentialVectorMean(\textcolor{red}{\textless type((treeNode))\ node\arginout,\textcolor{red}{\textless type((treeNode))\textgreater} host\arginout) Returns the mean vector of the vector tangential velocity averaged over all orbits.

double precision velocityTotalRootMeanSquared(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless type((treeNode))\textgreater} host\arginout) Returns the square root of the mean of the squared total velocity averaged over all orbits.

virialOrbitFixed

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((virialOrbitClass))\textgreater} destination\arginout) Perform a deep copy of the object.

class(virialDensityContrastClass) densityContrastDefinition() Returns a virialDensityContrast object describing the virial density contrast used to define this orbit class.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

type(keplerOrbit) orbit(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolo type((treeNode))\textgreater} host\arginout,\textcolor{red}{\textless logical\textgreater} acceptUnboundOrbits\argin) Returns an orbit object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision velocityTangentialMagnitudeMean(\textcolor{red}{\textless type((treeNode))\textgreater node\arginout,\textcolor{red}{\textless type((treeNode))\textgreater} host\arginout) Returns the mean of the magnitude of tangential velocity averaged over all orbits.

double precision, dimension(3) velocityTangentialVectorMean(\textcolor{red}{\textless type((treeNode))\ node\arginout,\textcolor{red}{\textless type((treeNode))\textgreater} host\arginout) Returns the mean vector of the vector tangential velocity averaged over all orbits.

double precision velocityTotalRootMeanSquared(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless type((treeNode))\textgreater} host\arginout) Returns the square root of the mean of the squared total velocity averaged over all orbits.

virialOrbitIsotropic

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((virialOrbitClass))\textgreater} destination\arginout) Perform a deep copy of the object.

class(virialDensityContrastClass) densityContrastDefinition() Returns a virialDensityContrast object describing the virial density contrast used to define this orbit class.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

type(keplerOrbit) orbit(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolo type((treeNode))\textgreater} host\arginout,\textcolor{red}{\textless logical\textgreater} acceptUnboundOrbits\argin) Returns an orbit object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision velocityTangentialMagnitudeMean(\textcolor{red}{\textless type((treeNode))\textgreater node\arginout,\textcolor{red}{\textless type((treeNode))\textgreater} host\arginout) Returns the mean of the magnitude of tangential velocity averaged over all orbits.

double precision, dimension(3) velocityTangentialVectorMean(\textcolor{red}{\textless type((treeNode))\ node\arginout,\textcolor{red}{\textless type((treeNode))\textgreater} host\arginout) Returns the mean vector of the vector tangential velocity averaged over all orbits.

double precision velocityTotalRootMeanSquared(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless type((treeNode))\textgreater} host\arginout) Returns the square root of the mean of the squared total velocity averaged over all orbits.

virialOrbitJiang2014

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((virialOrbitClass))\textgreater} destination\arginout) Perform a deep copy of the object.

class(virialDensityContrastClass) densityContrastDefinition() Returns a virialDensityContrast object describing the virial density contrast used to define this orbit class.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

type(keplerOrbit) orbit(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolo type((treeNode))\textgreater} host\arginout,\textcolor{red}{\textless logical\textgreater} acceptUnboundOrbits\argin) Returns an orbit object.

<void> parametersSelect(<double> massHost→, <double> massSatellite→, <integer> i←, <integer> j←) Select the parameter set to use for this satellite/host pairing.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_- size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision velocityTangentialMagnitudeMean(\textcolor{red}{\textless type((treeNode))\textgreater node\arginout,\textcolor{red}{\textless type((treeNode))\textgreater} host\arginout) Returns the mean of the magnitude of tangential velocity averaged over all orbits.

double precision, dimension(3) velocityTangentialVectorMean(\textcolor{red}{\textless type((treeNode))\ node\arginout,\textcolor{red}{\textless type((treeNode))\textgreater} host\arginout) Returns the mean vector of the vector tangential velocity averaged over all orbits.

double precision velocityTotalRootMeanSquared(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolor{red}{\textless type((treeNode))\textgreater} host\arginout) Returns the square root of the mean of the squared total velocity averaged over all orbits.

virialOrbitSpinCorrelated

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\ character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((virialOrbitClass))\textgreater} destination\arginout) Perform a deep copy of the object.

class(virialDensityContrastClass) densityContrastDefinition() Returns a virialDensityContrast object describing the virial density contrast used to define this orbit class.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

type(keplerOrbit) orbit(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolo
type((treeNode))\textgreater} host\arginout,\textcolor{red}{\textless logical\textgreater}
acceptUnboundOrbits\argin) Returns an orbit object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision velocityTangentialMagnitudeMean(\textcolor{red}{\textless type((treeNode))\textgreater
node\arginout,\textcolor{red}{\textless type((treeNode))\textgreater} host\arginout)
Returns the mean of the magnitude of tangential velocity averaged over all orbits.

double precision, dimension(3) velocityTangentialVectorMean(\textcolor{red}{\textless type((treeNode))\
node\arginout,\textcolor{red}{\textless type((treeNode))\textgreater} host\arginout)
Returns the mean vector of the vector tangential velocity averaged over all orbits.

double precision velocityTotalRootMeanSquared(\textcolor{red}{\textless type((treeNode))\textgreater}
node\arginout,\textcolor{red}{\textless type((treeNode))\textgreater} host\arginout)
Returns the square root of the mean of the squared total velocity averaged over all orbits.

virialOrbitWetzel2010

void allowedParameters(\textcolor{red}{\textless type((varying_string))\textgreater} allowedParameters\
character((len=*))\textgreater} sourceName\argin) Return a list of parameter names allowed for this object.

void autoHook() Insert any event hooks required by this object.

void deepCopy(\textcolor{red}{\textless class((virialOrbitClass))\textgreater} destination\arginout)
Perform a deep copy of the object.

class(virialDensityContrastClass) densityContrastDefinition() Returns a virialDensityContrast object describing the virial density contrast used to define this orbit class.

void descriptor(\textcolor{red}{\textless type((inputParameters))\textgreater} descriptor\arginout,\tex
logical\textgreater} includeMethod\argin) Return an input parameter list descriptor which could be used to recreate this object.

type(varying_string) hashedDescriptor(\textcolor{red}{\textless logical\textgreater} includeSourceDiges
Return a hash of the descriptor for this object, optionally include the source code digest in the hash.

<logical> isDefault() Return true if this is the default object of this class.

type(varying_string) objectType() Return the type of the object.

type(keplerOrbit) orbit(\textcolor{red}{\textless type((treeNode))\textgreater} node\arginout,\textcolo
type((treeNode))\textgreater} host\arginout,\textcolor{red}{\textless logical\textgreater}
acceptUnboundOrbits\argin) Returns an orbit object.

<integer> referenceCountDecrement() Decrement the reference count to this object and return the new reference count.

<void> referenceCountIncrement() Increment the reference count to this object.

<void> referenceCountReset() Reset the reference count to this object to 0.

void stateRestore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textl
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Restore the state of this object from file.

void stateStore(\textcolor{red}{\textless integer\textgreater} stateFile\argin,\textcolor{red}{\textles
type((fgsl_file))\textgreater} fgslStateFile\argin,\textcolor{red}{\textless integer((c_-
size_t))\textgreater} stateOperationID\argin) Store the state of this object to file.

double precision velocityTangentialMagnitudeMean(\textcolor{red}{\textless type((treeNode))\textgreater
node\arginout,\textcolor{red}{\textless type((treeNode))\textgreater} host\arginout)
Returns the mean of the magnitude of tangential velocity averaged over all orbits.

double precision, dimension(3) velocityTangentialVectorMean(\textcolor{red}{\textless type((treeNode))\
node\arginout,\textcolor{red}{\textless type((treeNode))\textgreater} host\arginout)
Returns the mean vector of the vector tangential velocity averaged over all orbits.

double precision velocityTotalRootMeanSquared(\textcolor{red}{\textless type((treeNode))\textgreater}
node\arginout,\textcolor{red}{\textless type((treeNode))\textgreater} host\arginout)
Returns the square root of the mean of the squared total velocity averaged over all orbits.

window

<logical> pointIncluded(<double(3)> point→) Return true if the given point lives inside the MANGLE window.

<void> read(<character(len=*)> fileName→) Read the specified MANGLE polygon file.

16. Adding New Methods

16.1. Code Directives

Galacticus is designed to be flexible and extensible, allowing you to add new methods and functionality without having to hack the code extensively. To achieve this it makes much use of embedded code directives which, for example, explain to the build system how a particular subroutine or function connects into the Galacticus code. Such code directives are indicated by lines beginning with `!#`, and take the form of short blocks of XML. For example, a typical code directive might look like:

```
!# <accretionDisksMethod>
!#  <unitName>Accretion_Disks_Shakura_Sunyaev_Initialize</unitName>
!# </accretionDisksMethod>
```

This directive would typically appear just prior to a subroutine which initializes the Shakura-Sunyaev accretion disk module (it could appear anywhere throughout that module, but it makes sense to keep it close to the subroutine that it references). The `accretionDisksMethod` tag explains to the Galacticus build system that this module contains an implementation of black hole accretion disks. The `unitName` tag specifies the name of a program unit which (in this case) should be called to initialize this accretion disk implementation. The build system will then insert appropriate `use` and `call` statements into the Galacticus code such that this routine will be called if and when accretion disks are required by Galacticus.

16.2. Identifying Components and Mass Types

Many functions can be applied to different components or groups of components and to different types of mass within a node. In general, these functions make use of a set of label defined in the `Galactic_Structure_Options` module. Components are identified by a `componentType` label which can take on the following values:

`componentTypeAll` All components are matched;

`componentTypeDisk` Only disk components are matched;

`componentTypeSpheroid` Only spheroid components are matched.

`componentTypeBlackHole` Only black hole components are matched.

`componentTypeHotHalo` Only hot halo components are matched.

`componentTypeDarkHalo` Only dark matter halo components are matched.

Types of mass are identified by a `massType` which can take one of the following values:

`massTypeAll` All mass is included;

`massTypeDark` Only dark matter is included;

`massTypeBaryonic` Only baryonic mass is included;

`massTypeGalactic` Only galactic mass is included.

`massTypeGaseous` Only gaseous mass is included.

`massTypeStellar` Only stellar mass is included.

`massTypeBlackHole` Only black hole mass is included.

16.3. Components

This section describes the internal structure of node components, and how a component is implemented.

16.3.1. Component Structure

Each node in the merger tree consists of an arbitrary number of "components", each of which can actually be an array, allowing multiple components of a given class. Each component represents a specific class of object, which could be a dark matter halo, a galactic disk or a black hole etc. A component of each class may be of one or more differnt implementations of that component class. Component classes are extensions of the `nodeComponent` base class, while each implementation is an extension of its component class (or, sometimes, of another implementation of that same class). Each component implementation type consists of a set of data[1], representing the properties (mass, size etc.) of the component, along with the rates of change (and ODE solver tolerances) for any properties which are evolvable. Additionally, each component contains a large number of methods (functions) which can be used to access its properties, query its interfaces and which are used internally to perform ODE evolution, output etc. The `nodeComponent` base class and all classes derived from it are built automatically by `Galacticus::Build::Components` at compile time (take a look in `work/build/objects.nodes.components.Inc` if you want to see the generated code).

16.3.2. Extending Components

It is possible to create a component which extends an existing component (see the discussion of the `extends` element in §16.3.3). This capability is intended to allow new properties to be added to a component without having to create a whole new copy of the component. It is *not* intended to allow changes in the way in which the component is evolved through the halo hierarchy. (With the exception that rules to describe how the newly added properties will evolve through the halo hierarchy can be added of course.)

A simple example of this extension capability can be found in the `scaleShape` dark matter profile component (§11.13.3), which extends the `scale` dark matter profile component (§11.13.1). In this case, the `scaleShape` component adds a new property, `shape`, and specifies how it is to be initialized, evolved, output, and change by node promotion events. It *does not* affect how the `scale` property, inherited from the `scale` dark matter profile component, is evolved.

[1] Data objects in components can be real, integer, boolean or of derived type. For derived types, currently `history`, `abundances`, `chemicals`, and `keplerOrbit` are supported. Adding additional derived types is possible, providing that the type supports the required methods for output, serialization, etc. Data objects can currently be scalar or rank-1 arrays.

16.3.3. Implementing a New Component

Implementing a new component involves writing some modules and functions which contain a definition of the component and, if necessary, handle initialization, creation, evolution, and responses to any events. Frequently, the easiest way to make a new component is to copy a previously existing one and modify it as needed. Details of the various functions that component modules must perform are given below. By convention, a component's implementation is split into three or four files, although some components might not need all of these files. These files are named as follows (with `<component>` acting as a placeholder for the name of the component in question):

`objects.nodes.components.<component>.F90` The primary file which describes the component and its properties, and which contains functions that manipulate the component as it evolves through a merger tree (ODE rates, behavior during mergers, etc.);

`objects.nodes.components.<component>.bound_functions.F90` Contains functions which will be bound to the component object (i.e. the `nodeComponent<Class><Implementation>` class), and so will be available as type bound procedures. Generally, these functions will include any which get or set values of properties in the component, those which return information about its internal state (such as the density at some position in the component; see §16.4.3), and any other functions which we may want to be overridden by extensions to the component.

`objects.nodes.components.<component>.data.F90` Contains any data which may need to be shared between the above two files. This might contain parameters which control some property of the component that is the same for all instances (e.g. if spheroids are modelled as Sérsic profiles all with the same value of the Sérsic index, that value might be placed into this file).

`objects.nodes.components.<component>.structure.F90` Contains any functions which implement the structure (e.g. density, rotation curve) of the component and which cannot be placed in `objects.nodes.components.<c` `functions.Inc` due to dependencies on modules which in turn depend on the `Galacticus_Nodes` module.

In general, `objects.nodes.components.<component>.F90` is the place for the component definition and functions which process the component during tree evolution (including output), while `objects.nodes.components.<compone` `functions.Inc` is intended for functions which record or report the internal state of the component.

Component Definition

Component definition itself takes the form of an embedded XML document. The following example illustrates such a document:

```
!# <component>
!#  <class>disk</class>
!#  <name>exponential</name>
!#  <isDefault>yes</isDefault>
!#  <properties>
!#   <property>
!#     <name>isInitialized</name>
!#     <type>logical</type>
!#     <rank>0</rank>
!#     <attributes isSettable="true" isGettable="true" isEvolvable="false" />
!#   </property>
!#   <property>
!#     <name>massStellar</name>
```

```
!#     <type>real</type>
!#     <rank>0</rank>
!#     <attributes isSettable="true" isGettable="true" isEvolvable="true" />
!#     <output unitsInSI="massSolar" comment="Mass of stars in the exponential disk."/>
!#   </property>
!#   <property>
!#     <name>abundancesStellar</name>
!#     <type>abundances</type>
!#     <rank>0</rank>
!#     <attributes isSettable="true" isGettable="true" isEvolvable="true" />
!#     <output unitsInSI="massSolar" comment="Mass of metals in the stellar phase of the exponential d
!#   </property>
!#   <property>
!#     <name>massGas</name>
!#     <type>real</type>
!#     <rank>0</rank>
!#     <attributes isSettable="true" isGettable="true" isEvolvable="true" createIfNeeded="true" makeGe
!#     <output unitsInSI="massSolar" comment="Mass of gas in the exponential disk."/>
!#   </property>
!#   <property>
!#     <name>coolingMass</name>
!#     <attributes isSettable="false" isGettable="false" isEvolvable="true" isDeferred="rate" bindsTo=
!#     <type>real</type>
!#     <rank>0</rank>
!#     <isVirtual>yes</isVirtual>
!#   </property>
!#   <property>
!#     <name>halfMassRadius</name>
!#     <attributes isSettable="false" isGettable="true" isEvolvable="false" />
!#     <type>real</type>
!#     <rank>0</rank>
!#     <isVirtual>yes</isVirtual>
!#     <getFunction>Node_Component_Disk_Exponential_Half_Mass_Radius</getFunction>
!#   </property>
!#   <property>
!#     <name>luminositiesStellar</name>
!#     <type>real</type>
!#     <rank>1</rank>
!#     <attributes isSettable="true" isGettable="true" isEvolvable="true" />
!#     <classDefault modules="Stellar_Population_Properties_Luminosities" count="Stellar_Population_Lu
!#     <output labels="':'//Stellar_Population_Luminosities_Name({i})" count="Stellar_Population_Lumin
!#   </property>
!#  </properties>
!#  <bindings>
!#   <binding method="attachPipes" function="Node_Component_Disk_Exponential_Attach_Pipes" type="void'
!#  </bindings>
!#  <functions>objects.nodes.components.disk.exponential.custom_methods.inc</functions>
!# </component>
```

The elements of this document have the following meaning:

`class` *[Required]* Specifies the component class of which this is an implementation.

`name` *[Required]* Specifies the name of this specific implementation.

`extends` *[Optional]* If present, this element must contain `class` and `name` elements which specify the type of component which should be extended. The component then automatically inherits all properties and type-bound functions of the extended type.

`isDefault` *[Required]* Specifies whether or not this should be the default implementation of this class. Note that only one implementation of each class can be declared to be the default. If no implementation of a given class is declared to be the default then the (automatically generated) `null` implementation will be made the default.

`properties` *[Optional]* Contains an array of `property` elements which specify the properties of this implementation. Each member `property` has the following structure:

`name` *[Required]* The name of the property.

`type` *[Required]* The type (one of `real`, `integer`, `logical`, `history`, `abundances`, `chemicals`, or `keplerOrbit` at present) of the property.

`rank` *[Required]* The rank of this property (currently `0` for a scalar or `1` for a 1-D array).

`attributes` *[Required]* Attributes of this property:

`isSettable` If `true` then the value of this property can be set directory.

`isGettable` If `true` then the value of this property can be got directory.

`isEvolvable` If `true` this property evolves as part of the Galacticus ODE system.

`createIfNeeded` If `true` then any attempt to get, set, or adjust the rate of this property will cause the component to be created if it does not already exist. This is useful if the component should be created in response to mass transfer from some other component for example.

`makeGeneric` If `true` then any `rate` method for this property will have a version created which binds to the base `nodeComponent` class. This version is suitable for attaching to deferred rate functions of components of another class. For example, the disk gas mass rate function is made generic, and then attached to the deferred cooling rate of the hot halo using:

```
call hotHalo%hotHaloCoolingMassRateFunction(DiskExponentialMassGasRateGeneric)
```

`isDeferred` Contains a "`:`" separated list which can contain `get`, `set`, and `rate`. The methods present in this list will not have functions bound to them at compile time. Instead a function will be created which allows a function to be bound to these methods at run time. For example:

```
call myComponent%massFunction    (My_Component_Mass_Get_Function)
call myComponent%massSetFunction (My_Component_Mass_Set_Function)
call myComponent%massRateFunction(My_Component_Mass_Rate_Function)
```

Additionally, a method is created which returns true or false depending on whether the method has been attached to a function yet, e.g.

```
myComponent%massIsAttached    ()
myComponent%massSetIsAttached ()
myComponent%massRateIsAttached()
```

`bindsTo` Specifies to which level in the class hierarchy set, get and rate methods should be bound. Normally, these are bound to the component implementation itself. However, it can be useful to specify a binding of "`top`" to bind to the base `nodeComponent` class to make these methods interoperable with properties of other classes (see the discussion of the `makeGeneric` element above).

`output` *[Optional]* If present, the property will be included in the Galacticus output file. The following attributes control the details of that output:

`unitsInSI` The units of the output quantity in the SI system.

`comment` A comment to be included with the HDF5 dataset for this property.

`condition` A statement which must evaluate to `true` or `false` and which will be used to determine if the property will be output. The present output time for is available as `time`. In the case of an array property the construct "`{i}`" can be used to pass the index of the element for which the condition should be evaluated.

`modules` A comma-separated list of any modules required to perform the output (e.g. modules which contain functions or values that are used).

Additional attributes are required for array properties:

`labels` This can be an array, declared as "`[`L_1`,...,`L_N`]`", specifying the suffix to be added to the property name for each component of the array in the output, or a function which returns the suffix. In the case of a function the construct "`{i}`" can be used to pass the index of the element for which the suffix is required.

`count` A statement which evaluates the the number of elements to be output (i.e. the length of the array).

`isVirtual` *[Optional]* If present and set to "`yes`", this property is a virtual property. A virtual property has no data associated with it and must supply its own functions for getting, setting and adjusting its rate of change (if allowed by the property's attributes). Virtual properties are used for quantities which are derived from actual properties of the component implementation (for example, a star formation rate could be a virtual property if it is derived from an actual gas mass property) or for adjusting the rates of several actual properties simultaneously.

`getFunction` *[Optional]* Specifies the function to be used for getting the value of the property, overriding the default get function. The function must be included in the `Galacticus_Nodes` module by use of the `functions` element described below. Note that this function, by virtue of its priveleged access to the itnernal structure of node components, can access the value of the data associated with the propery using:

```
myComponent%<property>Data%value
```

`setFunction` *[Optional]* The same as `getFunction` but defines a function to set the value of the property.

`classDefault` *[Optional]* Specifies the default value for this property if the component class has not been created (i.e. has no specific implementation yet). The content of this element gives the default value (which can be a scalar, an array, a function, etc.). Additional, optional attributes control the use of this element:

`modules` Specifies a comma-separated list of modules which are required to set the default values (e.g. modules which contain the value or function to be used).

`count` For array properties whose size is not known at compile-time, it is possible to specify a function which will return the appropriate size of the array at run-time. The scalar default value given in the `classDefault` element will then be replicated the appropriate number of times.

`bindings` *[Optional]* Contains an array of `binding` elements which specify functions to bind to this implementation. Each member `binding` has the following structure:

- `method` The name of the bound method, such that the function can be accessed using

  ```
  myComponent%<method>(...)
  ```

- `function` The function to which the method should be bound. (This function must be included in the `Galacticus_Nodes` module by use of the `functions` element described below.
- `type` The type of function.
- `bindsTo` Specifies where this method should be bound. "`component`" specifies binding to the specific implementation of this component class, "`componentClass`" specifies binding to the component class, while "`top`" specifies binding to the base `nodeComponent` class.

`functions` *[Optional]* Contains the name of a file which will be included into the `Galacticus_Nodes` module. This file can contain functions which will be bound to this implementation. By virtue of being included in the `Galacticus_Nodes` module these functions have privelged access to the internal structure of all node component objects.

Component Initialization

Initialization of a component module (if necessary, for example, to read parameters or allocate workspace) can occur at a number of different points in the execution of Galacticus. Providing initialization occurs in advance of any calculations then any point is acceptable. One possibility is simply to call an initialization function at the head of all functions defined in the component module. This initialization function should return immediately if it has already been called (to avoid duplicate initialization). Another option is to use the `mergerTreePreTreeConstructionTask` event (see §16.4.3) to perform initialization just before merger trees are constructed (the initialization function must again return immediately if it has been previously called).

Optionally, a component may include a `mergerTreeEvolveThreadInitialize` directive, which gives the name of a subroutine in its `unitName` element. The routine specified by `mergerTreeEvolveThreadInitialize` is called by all threads prior to merger tree evolution, and can therefore be used to perform any "per thread" initialization. Note that this routine will be called many times during a given Galacticus run—it is the responsibility of the routine to ensure that it performs any initialization only once.

Component Access, Creation and Destruction

When a node is created, it initially contains no components. A component must therefore create itself on the fly as needed. Typically, a component is first created when an attempt is made to set a property value, or to adjust the rate of change of a property value or in response to some event (e.g. a satellite component may be created in response to a node merging with a larger node). Requests for property values frequently *do not* require that the component exist, as a zero value can often be returned instead[2].

To access a component from a node, use:

```
myComponent => thisNode%<class>([instance=<N>,autoCreate=<create>])
```

where `class` is the component class required, the optional `instance` argument requests a specific instance of the component (relevant if the node contains more than one of a particular component, e.g. if it contains two supermassive black holes for example; if no `instance` is specified the first instance will be returned), and the `autoCreate` option specifies whether or not the component should be automatically created (assuming it does not already exist). `autoCreate=true` should be used to create components initially.

A component of a node can be destroyed using:

[2] Or some other value if a `classDefault` has been specified (see §16.3.3).

```
call thisNode%<class>Destroy()
```

Component Methods

Component implementations optionally provide functions to get and set their properties (and to set the rate of change of evolvable properties) so that other components and functions within Galacticus to can interact with them in a way that is independent of the specific component implementation chosen. To permit this, Galacticus creates functions for each property to access it in all permitted ways. For example, the `exponential` implementation of the `disk` component class has a "`massStellar`" property defined by:

```
<method>
  <name>massStellar</name>
  <type>real</type>
  <rank>0</rank>
  <attributes isSettable="true" isGettable="true" isEvolvable="true" />
</method>
```

This causes Galacticus to define several functions bound to the `nodeComponentDisk` class:

`massStellarIsSettable` Returns `true` if this property is settable;

`massStellarIsGettable` Returns `true` if this property is gettable;

`massStellarSet` Sets the value of this property to the supplied argument;

`massStellarGet` Gets the value of this property;

`massStellarRate` Cumulates its argument to the rate of change of this property;

`massStellarScale` Sets the absolute scale for this property used in ODE error control;

along with several others used internally for output, serialization etc.

Component Evolution

All component properties which have an `isEvolvable` attribute set to `true` are included in Galacticus's ODE solver as the node is evolved forward in time. As described in §16.3.3, Galacticus will create two functions that permit the rate of change of a property adjusted and for the absolute scale used in ODE error control to be set.

A "rate compute" function should be defined to perform any calculations necessary to determine the rate of change of the property and adjust the rate appropriately. Below is an example of the rate compute subroutine for the stellar mass property of the exponential disk component, with only the basic structure shown:

```
!# <rateComputeTask>
!#  <unitName>Node_Component_Disk_Exponential_Rate_Compute</unitName>
!# </rateComputeTask>
subroutine Node_Component_Disk_Exponential_Rate_Compute(node,&
   &odeConverged,interrupt,interruptProcedure,propertyType)
  implicit none
  type     (treeNode                ), pointer , intent(inout) :: node
  logical                            , intent(in    ) :: &
     &odeConverged
```

```
    logical                                     , intent(inout) :: interrupt
    procedure(                        ), pointer, intent(inout) :: &
       &interruptProcedure
    integer                                     , intent(in    ) :: &
       &propertyType
    class     (nodeComponentDisk      ), pointer                :: disk

    ! Get the disk and check that it is of our class.
    disk => node%disk()
    select type (disk)
    class is (nodeComponentDiskExponential)
      ...
      call disk%massStellarRate(stellarMassRate)
      ...
    end select
    return
  end subroutine Node_Component_Disk_Exponential_Rate_Comput
```

Here, we get the disk component and check that it of the `exponential` variety. If it is, we compute the rates of change for one or more properties and then adjust their rates appropriately. If multiple instances of a component are used then the rate compute function should loop over all instances and adjust rates appropriately. The `propertyType` argument will be set either to `propertyTypeActive` or `propertyTypeInactive`, and indicates if rates should be computed for active or inactive variables. (Rates for active variables can be computed and set even if inactive variables are requested, and vice versa. They will simply be ignored, and so this merely wastes compute time.) The `odeConverged` variable indicates if the ODE solver has reached convergence—in some instances it is useful to trigger interrupts only once convergence is reached. If an interrupt occurs, `interrupt` should be set to `true`, and `interruptProcedure` should be set to point to the function to call on interrupt.

When evolving ODEs the ODE solver aims to keep the error on property i below

$$D_i = \epsilon_{\rm abs} s_i + \epsilon_{\rm rel} |y_i|, \tag{16.1}$$

where $epsilon_{\rm abs}$ =`[odeToleranceAbsolute]`, $epsilon_{\rm rel}$ =`[odeToleranceRelative]`, y_i is the value of property i and s_i is a scaling factor which controls the absolute tolerance for this property. By default, $s_i = 1$, but this can be changed for a component utilizing the `scaleSetTask` directive. This allows a function to be called in which the component sets suitable scale factors for each of its properties prior to any ODE evolution being carried out. This can be very useful, for example, in cases where two components are coupled. Consider a case where a disk is transferring material to a spheroid via a bar instability. If the disk is orders of magnitude more massive that the spheroid then the rate of mass transfer can be very high (i.e. $\dot{y}/y$ for the spheroid will be large). With just a relative tolerance (i.e. the $\epsilon_{\rm rel}|y_i|$ term) this would require very short timesteps for the spheroid. However, in such cases we don't care about such tiny tolerances for the spheroid (since it will grow to be substantially more massive). Therefore, it may be appropriate to set s_i to be equal to the sum of the disk and spheroid properties for example. The scale set directive and associated subroutine should follow this template:

```
!# <scaleSetTask>
!#  <unitName>Node_Component_Disk_Exponential_Scale_Set</unitName>
!# </scaleSetTask>
subroutine Node_Component_Disk_Exponential_Scale_Set(thisNode)
  implicit none
  type (treeNode         ), pointer, intent(inout) :: thisNode
```

```
    class(nodeComponentDisk), pointer                :: disk

    ! Get the disk component.
    disk => thisNode%disk()
    ! Check if an exponential disk component exists.
    select type (disk)
    class is (nodeComponentDiskExponential)
      ...
      call disk%massStellarScale(massScale)
      ...
    end select
    return
  end subroutine Node_Component_Disk_Exponential_Scale_Set
```

Sensible choices for the s_i factors can significantly speed-up execution of Galacticus.

Evolution Interrupts

It is often necessary to interrupt the smooth ODE evolution of a node in Galacticus. This can happen if, for example, a galaxy mergers with another galaxy (in which case the merger must be processed prior to further evolution) or if a component must be created before evolution can continue. The rate adjust and rate compute subroutines allow for interrupts to be flagged via their `interrupt` and `interruptProcedure` arguments. If an interrupt is required then `interrupt` should be set to true, while `interruptProcedure` should be set to point to a procedure which will handle the interrupt. Then, providing no other interrupt occurred earlier, the evolution will be stopped and the interrupt procedure called before evolution is continued.

An interrupt procedure should have the form:

```
subroutine My_Interrupt_Procedure(thisNode)
  implicit none
  type(treeNode), pointer, intent(inout) :: thisNode

  ! Do whatever needs to be done to handle the interrupt.

  return
end subroutine My_Interrupt_Procedure
```

16.4. Existing Method Types

16.4.1. Functions

Functions implement basic calculations (e.g. computing the power spectrum).

Accretion Disk Spectra

Additional implementations for accretion disk spectra are added using the `accretionDiskSpectra` class. The implementation should be placed in a file containing the directive:

```
!# <accretionDiskSpectra name="accretionDiskSpectraMyImplementation">
!# <description>A short description of the implementation.</description>
!# </accretionDiskSpectra>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `accretionDiskSpectraMyImplementation`. The file *must* define a type that extends the `accretionDiskSpectraClass` class (or extends another type which is itself an extension of the `accretionDiskSpectraClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`spectrum` Returns the spectrum (in units of $L_\odot$ Hz^{-1}) of the accretion disk at the given wavelength (in units of Å) for `node`. Must have the following interface:

```
double precision function myImplementationSpectrum(self,node,wavelength)
 class           (accretionDiskSpectraClass), intent(inout) :: self
 type            (treeNode                 ), intent(inout) :: node
 double precision                           , intent(in   ) :: wavelength
end double precision function myImplementationSpectrum
```

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
 class  (accretionDiskSpectraClass), intent(inout)           :: self
 logical                           , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (accretionDiskSpectraClass), intent(inout)           :: self
 type   (inputParameters          ), intent(inout)           :: descriptor
 logical                           , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(accretionDiskSpectraClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (accretionDiskSpectraClass), intent(inout) :: self
 integer                           , intent(in   ) :: stateFile
 type   (fgsl_file                 ), intent(in   ) :: fgslStateFile
 integer(c_size_t                  ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (accretionDiskSpectraClass)                          , intent(inout) :: self
 type     (varying_string           ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                    )                          , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(accretionDiskSpectraClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(accretionDiskSpectraClass), intent(inout) :: self
 class(accretionDiskSpectraClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (accretionDiskSpectraClass), intent(inout) :: self
 integer                           , intent(in   ) :: stateFile
 type   (fgsl_file                ), intent(in   ) :: fgslStateFile
 integer(c_size_t                 ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`accretionDiskSpectraHopkins2007` Accretion disk spectra using the model of Hopkins et al. [2007].

`accretionDiskSpectraFile` Accretion disk spectra are interpolated from tables read from file.

Accretion disks

Additional implementations for accretion disks are added using the `accretionDisks` class. The implementation should be placed in a file containing the directive:

```
!# <accretionDisks name="accretionDisksMyImplementation">
!# <description>A short description of the implementation.</description>
!# </accretionDisks>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all

functions required by the implementation after that line. Function names should begin with `accretionDisksMyImplementation`. The file *must* define a type that extends the `accretionDisksClass` class (or extends another type which is itself an extension of the `accretionDisksClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (accretionDisksClass), intent(inout)           :: self
  logical                     , intent(in   ), optional :: includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (accretionDisksClass), intent(inout)           :: self
 type   (inputParameters    ), intent(inout)           :: descriptor
 logical                     , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(accretionDisksClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (accretionDisksClass), intent(inout) :: self
 integer                     , intent(in   ) :: stateFile
 type   (fgsl_file          ), intent(in   ) :: fgslStateFile
 integer(c_size_t           ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (accretionDisksClass)                          , intent(inout) :: self
 type     (varying_string     ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*              )                          , intent(in   ) :: sourceName
end subroutine myImplementationAllowedParameters
```

`powerJet` Returns the power of the jet launched by the accretion disk in units of $M_\odot$ $(\mathrm{km/s})^2$ Gyr^{-1}. Must have the following interface:

```
double precision function myImplementationPowerJet(self,blackHole,accretionRateMass)
 class           (accretionDisksClass   ), intent(inout) :: self
 class           (nodeComponentBlackHole), intent(inout) :: blackHole
 double precision                        , intent(in   ) :: accretionRateMass
end double precision function myImplementationPowerJet
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(accretionDisksClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`efficiencyRadiative` Returns the radiative efficiency of the accretion disk. Must have the following interface:

```
double precision function myImplementationEfficiencyRadiative(self,blackHole,&
&  accretionRateMass)
 class           (accretionDisksClass   ), intent(inout) :: self
 class           (nodeComponentBlackHole), intent(inout) :: blackHole
 double precision                        , intent(in   ) :: accretionRateMass
end double precision function myImplementationEfficiencyRadiative
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(accretionDisksClass), intent(inout) :: self
 class(accretionDisksClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`rateSpinUp` Returns the spin-up rate of the black hole due to accretion from the accretion disk. Must have the following interface:

```
double precision function myImplementationRateSpinUp(self,blackHole,accretionRateMass)
 class           (accretionDisksClass   ), intent(inout) :: self
 class           (nodeComponentBlackHole), intent(inout) :: blackHole
 double precision                        , intent(in   ) :: accretionRateMass
end double precision function myImplementationRateSpinUp
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (accretionDisksClass), intent(inout) :: self
 integer                     , intent(in   ) :: stateFile
 type   (fgsl_file          ), intent(in   ) :: fgslStateFile
 integer(c_size_t           ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`accretionDisksSwitched` An accretion disk class in which accretion switches between thin-disk and ADAF modes.

`accretionDisksShakuraSunyaev` A Shakura and Sunyaev [1973] accretion disk class.

`accretionDisksEddingtonLimited` An accretion disk class in which accretion is always Eddington-limited.

`accretionDisksADAF` An ADAF accretion disk class.

Accretion Onto Halos

Additional implementations for accretion onto halos are added using the `accretionHalo` class. The implementation should be placed in a file containing the directive:

```
!# <accretionHalo name="accretionHaloMyImplementation">
!# <description>A short description of the implementation.</description>
!# </accretionHalo>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `accretionHaloMyImplementation`. The file *must* define a type that extends the `accretionHaloClass` class (or extends another type which is itself an extension of the `accretionHaloClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (accretionHaloClass), intent(inout)          :: self
  logical                    , intent(in   ), optional :: includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationDescriptor(self,descriptor,includeMethod)
  class  (accretionHaloClass), intent(inout)          :: self
  type   (inputParameters   ), intent(inout)          :: descriptor
  logical                    , intent(in   ), optional :: includeMethod
 end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAutoHook(self)
  class(accretionHaloClass), intent(inout) :: self
 end subroutine myImplementationAutoHook
```

`accretionRateChemicals` Returns the rate (in units of $M_\odot$ Gyr^{-1}) of accretion of chemicals from the IGM onto `node` in the given `accretionMode`. Must have the following interface:

```
 type(chemicalAbundances) function myImplementationAccretionRateChemicals(self,node,&
&  accretionMode)
  class  (accretionHaloClass), intent(inout) :: self
  type   (treeNode          ), intent(inout) :: node
  integer                    , intent(in   ) :: accretionMode
 end type(chemicalAbundances) function myImplementationAccretionRateChemicals
```

`accretedMassChemicals` Returns the mass of chemicals (in units of $M_\odot$) of accreted from the IGM onto `node` in the given `accretionMode`. Used to initialize nodes. Must have the following interface:

```
 type(chemicalAbundances) function myImplementationAccretedMassChemicals(self,node,&
&  accretionMode)
  class  (accretionHaloClass), intent(inout) :: self
  type   (treeNode          ), intent(inout) :: node
  integer                    , intent(in   ) :: accretionMode
 end type(chemicalAbundances) function myImplementationAccretedMassChemicals
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
  class  (accretionHaloClass), intent(inout) :: self
  integer                    , intent(in   ) :: stateFile
  type   (fgsl_file         ), intent(in   ) :: fgslStateFile
  integer(c_size_t          ), intent(in   ) :: stateOperationID
 end subroutine myImplementationStateStore
```

`accretionRate` Returns the rate (in units of $M_\odot$ Gyr^{-1}) of accretion of mass from the IGM onto `node` in the given `accretionMode`. Must have the following interface:

```
 double precision function myImplementationAccretionRate(self,node,accretionMode)
  class  (accretionHaloClass), intent(inout) :: self
  type   (treeNode          ), intent(inout) :: node
  integer                    , intent(in   ) :: accretionMode
 end double precision function myImplementationAccretionRate
```

`accretedMassMetals` Returns the mass of metals (in units of $M_\odot$) of accreted from the IGM onto `node` in the given `accretionMode`. Used to initialize nodes. Must have the following interface:

```
 type(abundances) function myImplementationAccretedMassMetals(self,node,accretionMode)
  class  (accretionHaloClass), intent(inout) :: self
  type   (treeNode          ), intent(inout) :: node
  integer                    , intent(in   ) :: accretionMode
 end type(abundances) function myImplementationAccretedMassMetals
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
  class    (accretionHaloClass)                          , intent(inout) :: self
  type     (varying_string    ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
  character(len=*             )                          , intent(in   ) :: sourceName
 end subroutine myImplementationAllowedParameters
```

`accretionRateMetals` Returns the rate (in units of $M_\odot$ Gyr^{-1}) of accretion of metals from the IGM onto `node` in the given `accretionMode`. Must have the following interface:

```
type(abundances) function myImplementationAccretionRateMetals(self,node,accretionMode)
 class  (accretionHaloClass), intent(inout) :: self
 type   (treeNode          ), intent(inout) :: node
 integer                    , intent(in   ) :: accretionMode
end type(abundances) function myImplementationAccretionRateMetals
```

`failedAccretedMass` Returns the mass (in units of $M_\odot$) of that failed to accrete from the IGM onto `node` in the given `accretionMode`. Used to initialize nodes. Must have the following interface:

```
double precision function myImplementationFailedAccretedMass(self,node,accretionMode)
 class  (accretionHaloClass), intent(inout) :: self
 type   (treeNode          ), intent(inout) :: node
 integer                    , intent(in   ) :: accretionMode
end double precision function myImplementationFailedAccretedMass
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(accretionHaloClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(accretionHaloClass), intent(inout) :: self
 class(accretionHaloClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (accretionHaloClass), intent(inout) :: self
 integer                    , intent(in   ) :: stateFile
 type   (fgsl_file         ), intent(in   ) :: fgslStateFile
 integer(c_size_t          ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

`failedAccretionRate` Returns the rate (in units of $M_\odot$ Gyr^{-1}) of failed accretion of mass from the IGM onto `node` in the given `accretionMode`. Must have the following interface:

```
double precision function myImplementationFailedAccretionRate(self,node,accretionMode)
 class  (accretionHaloClass), intent(inout) :: self
 type   (treeNode          ), intent(inout) :: node
 integer                    , intent(in   ) :: accretionMode
end double precision function myImplementationFailedAccretionRate
```

`accretedMass` Returns the mass (in units of $M_\odot$) of accreted from the IGM onto `node` in the given `accretionMode`. Used to initialize nodes. Must have the following interface:

```
double precision function myImplementationAccretedMass(self,node,accretionMode)
 class  (accretionHaloClass), intent(inout) :: self
 type   (treeNode          ), intent(inout) :: node
 integer                    , intent(in   ) :: accretionMode
end double precision function myImplementationAccretedMass
```

`branchHasBaryons` Returns `true` if this tree branch may accrete baryons, and `false` otherwise. Must have the following interface:

```
logical function myImplementationBranchHasBaryons(self,node)
 class(accretionHaloClass), intent(inout)         :: self
 type (treeNode          ), intent(inout), target :: node
end logical function myImplementationBranchHasBaryons
```

Existing implementations are:

`accretionHaloColdMode` Accretion onto halos using simple truncation to mimic the effects of reionization and accounting for cold mode accretion. See §12.2.3.

`accretionHaloNaozBarkana2007` Accretion onto halos using filtering mass of the IGM calculated from an equation from Naoz and Barkana [2007]. See §12.2.4.

`accretionHaloBertschinger` Accretion onto halos using simple truncation to mimic the effects of reionization, and the Bertschinger mass to define available mass.

`accretionHaloZero` Accretion onto halos assuming no accretion.

`accretionHaloSimple` Accretion onto halos using simple truncation to mimic the effects of reionization. See §12.2.1.

Halo Total Accretion Rates

Additional implementations for halo total accretion rates are added using the `accretionHaloTotal` class. The implementation should be placed in a file containing the directive:

```
!# <accretionHaloTotal name="accretionHaloTotalMyImplementation">
!# <description>A short description of the implementation.</description>
!# </accretionHaloTotal>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `accretionHaloTotalMyImplementation`. The file *must* define a type that extends the `accretionHaloTotalClass` class (or extends another type which is itself an extension of the `accretionHaloTotalClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (accretionHaloTotalClass), intent(inout)           :: self
  logical                         , intent(in   ), optional :: includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (accretionHaloTotalClass), intent(inout)           :: self
 type   (inputParameters        ), intent(inout)           :: descriptor
 logical                         , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(accretionHaloTotalClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (accretionHaloTotalClass), intent(inout) :: self
 integer                         , intent(in   ) :: stateFile
 type   (fgsl_file               ), intent(in   ) :: fgslStateFile
 integer(c_size_t                ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`accretionRate` Return the total accretion rate onto the given `node`. Must have the following interface:

```
double precision function myImplementationAccretionRate(self,node)
 class(accretionHaloTotalClass), intent(inout) :: self
 type (treeNode                ), intent(inout) :: node
end double precision function myImplementationAccretionRate
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (accretionHaloTotalClass)                          , intent(inout) :: self
 type     (varying_string         ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                  )                          , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(accretionHaloTotalClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(accretionHaloTotalClass), intent(inout) :: self
 class(accretionHaloTotalClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (accretionHaloTotalClass), intent(inout) :: self
 integer                         , intent(in   ) :: stateFile
 type   (fgsl_file              ), intent(in   ) :: fgslStateFile
 integer(c_size_t               ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

`accretedMass` Return the total accreted mass in the given `node`. Must have the following interface:

```
double precision function myImplementationAccretedMass(self,node)
 class(accretionHaloTotalClass), intent(inout) :: self
 type (treeNode               ), intent(inout) :: node
end double precision function myImplementationAccretedMass
```

Existing implementations are:

`accretionHaloTotalBertschinger` A halo total accretion class which assumes the accretion corresponds to the basic mass. See §12.3.2.

`accretionHaloTotalSimple` A halo total accretion class which assumes the accretion corresponds to the basic mass. See §12.3.1.

Atomic cross sections for photo-ionization.

Additional implementations for atomic cross sections for photo-ionization. are added using the `atomicCrossSectionIonizati` class. The implementation should be placed in a file containing the directive:

```
!# <atomicCrossSectionIonizationPhoto name="atomicCrossSectionIonizationPhotoMyImplementation">
!# <description>A short description of the implementation.</description>
!# </atomicCrossSectionIonizationPhoto>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `atomicCrossSectionIonizationPhotoMyImplementation`. The file *must* define a type that extends the `atomicCrossSectionIonizationPhotoClass` class (or extends another type which is itself an extension of the `atomicCrossSectionIonizationPhotoClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (atomicCrossSectionIonizationPhotoClass), intent(inout)          :: self
  logical                                        , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationDescriptor(self,descriptor,includeMethod)
  class  (atomicCrossSectionIonizationPhotoClass), intent(inout)          :: self
  type   (inputParameters                       ), intent(inout)          :: descriptor
  logical                                        , intent(in   ), optional :: &
&  includeMethod
 end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAutoHook(self)
  class(atomicCrossSectionIonizationPhotoClass), intent(inout) :: self
 end subroutine myImplementationAutoHook
```

`crossSection` Returns the cross-section for photoionization (in units of cm^2) for a given atom in a given ionization state at the specified `wavelength` (given in units of Å). Must have the following interface:

```
 double precision function myImplementationCrossSection(self,atomicNumber, &
&  ionizationState, shellNumber,wavelength)
  class           (atomicCrossSectionIonizationPhotoClass), intent(inout) :: self
  integer                                                 , intent(in   ) :: &
&  atomicNumber, ionizationState, shellNumber
  double precision                                        , intent(in   ) :: wavelength
 end double precision function myImplementationCrossSection
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
  class  (atomicCrossSectionIonizationPhotoClass), intent(inout) :: self
  integer                                        , intent(in   ) :: stateFile
  type   (fgsl_file                             ), intent(in   ) :: fgslStateFile
  integer(c_size_t                              ), intent(in   ) :: stateOperationID
 end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
  class    (atomicCrossSectionIonizationPhotoClass)                        , intent(&
&  inout) :: self
  type     (varying_string                        ), allocatable, dimension(:), intent(&
&  inout) :: allowedParameters
  character(len=*                                 )                        , intent(&
&  in   ) :: sourceName
 end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(atomicCrossSectionIonizationPhotoClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(atomicCrossSectionIonizationPhotoClass), intent(inout) :: self
 class(atomicCrossSectionIonizationPhotoClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (atomicCrossSectionIonizationPhotoClass), intent(inout) :: self
 integer                                        , intent(in   ) :: stateFile
 type   (fgsl_file                              ), intent(in   ) :: fgslStateFile
 integer(c_size_t                               ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`atomicCrossSectionIonizationPhotoVerner` An atomic photoionization cross section class based on (`phfit2.f`) written by D. A. Verner (Version 2. March 25, 1996.

Atomic Collisional Excitation

Additional implementations for atomic collisional excitation are added using the `atomicExcitationRateCollisional` class. The implementation should be placed in a file containing the directive:

```
!# <atomicExcitationRateCollisional name="atomicExcitationRateCollisionalMyImplementation">
!# <description>A short description of the implementation.</description>
!# </atomicExcitationRateCollisional>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `atomicExcitationRateCollisionalMyImplementation`. The file *must* define a type that extends the `atomicExcitationRateCollisionalClass` class (or extends another type which is itself an extension of the `atomicExcitationRateCollisionalClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (atomicExcitationRateCollisionalClass), intent(inout)          :: self
  logical                                      , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationDescriptor(self,descriptor,includeMethod)
  class  (atomicExcitationRateCollisionalClass), intent(inout)          :: self
  type   (inputParameters                     ), intent(inout)          :: descriptor
  logical                                      , intent(in   ), optional :: &
&  includeMethod
 end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAutoHook(self)
  class(atomicExcitationRateCollisionalClass), intent(inout) :: self
 end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
  class  (atomicExcitationRateCollisionalClass), intent(inout) :: self
  integer                                      , intent(in   ) :: stateFile
  type   (fgsl_file                           ), intent(in   ) :: fgslStateFile
  integer(c_size_t                            ), intent(in   ) :: stateOperationID
 end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
  class    (atomicExcitationRateCollisionalClass)                          , intent(&
&  inout) :: self
  type     (varying_string                      ), allocatable, dimension(:), intent(&
&  inout) :: allowedParameters
  character(len=*                               )                          , intent(in &
&    ) :: sourceName
 end subroutine myImplementationAllowedParameters
```

`coolingRate` Return the collisional excitation cooling rate , in units of $\mathrm{J/m^3/s}$, for ion of given `atomicNumber` and `electronNumber` at temperature `T` (in Kelvin). Must have the following interface:

```
 double precision function myImplementationCoolingRate(self,atomicNumber, electronNumber&
&  ,temperature)
  class           (atomicExcitationRateCollisionalClass), intent(inout) :: self
  integer                                               , intent(in   ) :: atomicNumber,&
&   electronNumber
  double precision                                      , intent(in   ) :: temperature
 end double precision function myImplementationCoolingRate
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(atomicExcitationRateCollisionalClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(atomicExcitationRateCollisionalClass), intent(inout) :: self
 class(atomicExcitationRateCollisionalClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (atomicExcitationRateCollisionalClass), intent(inout) :: self
 integer                                      , intent(in   ) :: stateFile
 type   (fgsl_file                           ), intent(in   ) :: fgslStateFile
 integer(c_size_t                            ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`atomicExcitationRateCollisionalScholzWalters1991` Atomic collisional excitation using the fitting functions of Scholz and Walters [1991].

Atomic ionization potentials.

Additional implementations for atomic ionization potentials. are added using the `atomicIonizationPotential` class. The implementation should be placed in a file containing the directive:

```
!# <atomicIonizationPotential name="atomicIonizationPotentialMyImplementation">
!# <description>A short description of the implementation.</description>
!# </atomicIonizationPotential>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `atomicIonizationPotentialMyImplementation`. The file *must* define a type that extends the `atomicIonizationPotentialClass` class (or extends another type which is itself an extension of the `atomicIonizationPotentialClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (atomicIonizationPotentialClass), intent(inout)          :: self
  logical                                , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (atomicIonizationPotentialClass), intent(inout)          :: self
 type   (inputParameters               ), intent(inout)          :: descriptor
 logical                                , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(atomicIonizationPotentialClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`potential` Returns the ionization potential (in units of eV) for a given atom in a given ionization state. Must have the following interface:

```
double precision function myImplementationPotential(self,atomicNumber, electronNumber)
 class  (atomicIonizationPotentialClass), intent(inout) :: self
 integer                                , intent(in   ) :: atomicNumber, electronNumber
end double precision function myImplementationPotential
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (atomicIonizationPotentialClass), intent(inout) :: self
 integer                                , intent(in   ) :: stateFile
 type   (fgsl_file                     ), intent(in   ) :: fgslStateFile
 integer(c_size_t                      ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (atomicIonizationPotentialClass)                          , intent(inout) ::&
&   self
 type     (varying_string               ), allocatable, dimension(:), intent(inout) ::&
&   allowedParameters
 character(len=*                        )                          , intent(in   ) ::&
&   sourceName
 end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(atomicIonizationPotentialClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(atomicIonizationPotentialClass), intent(inout) :: self
 class(atomicIonizationPotentialClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (atomicIonizationPotentialClass), intent(inout) :: self
 integer                                , intent(in   ) :: stateFile
 type   (fgsl_file                      ), intent(in   ) :: fgslStateFile
 integer(c_size_t                       ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`atomicIonizationPotentialVerner` Implements an atomic ionization potential class, which provides potentials for all ionization stages of all atoms from H to Zn using data taken from Dima Verner's code.

Atomic Collisional Ionization

Additional implementations for atomic collisional ionization are added using the `atomicIonizationRateCollisional` class. The implementation should be placed in a file containing the directive:

```
!# <atomicIonizationRateCollisional name="atomicIonizationRateCollisionalMyImplementation">
!# <description>A short description of the implementation.</description>
!# </atomicIonizationRateCollisional>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `atomicIonizationRateCollisionalMyImplementation`. The file *must* define a type that extends the `atomicIonizationRateCollisionalClass` class (or extends another type which is itself an extension of the `atomicIonizationRateCollisionalClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (atomicIonizationRateCollisionalClass), intent(inout)          :: self
```

```
  logical                                     , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationDescriptor(self,descriptor,includeMethod)
  class  (atomicIonizationRateCollisionalClass), intent(inout)           :: self
  type   (inputParameters                     ), intent(inout)           :: descriptor
  logical                                      , intent(in   ), optional :: &
&  includeMethod
 end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAutoHook(self)
  class(atomicIonizationRateCollisionalClass), intent(inout) :: self
 end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
  class  (atomicIonizationRateCollisionalClass), intent(inout) :: self
  integer                                      , intent(in   ) :: stateFile
  type   (fgsl_file                           ), intent(in   ) :: fgslStateFile
  integer(c_size_t                            ), intent(in   ) :: stateOperationID
 end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
  class    (atomicIonizationRateCollisionalClass)                          , intent(&
&  inout) :: self
  type     (varying_string                      ), allocatable, dimension(:), intent(&
&  inout) :: allowedParameters
  character(len=*                               )                          , intent(in &
&    ) :: sourceName
 end subroutine myImplementationAllowedParameters
```

`rate` Returns the radiative recombination rate. Must have the following interface:

```
 double precision function myImplementationRate(self,atomicNumber, ionizationState,&
&  temperature)
  class           (atomicIonizationRateCollisionalClass), intent(inout) :: self
  integer                                               , intent(in   ) :: atomicNumber,&
&   ionizationState
  double precision                                      , intent(in   ) :: temperature
 end double precision function myImplementationRate
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(atomicIonizationRateCollisionalClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(atomicIonizationRateCollisionalClass), intent(inout) :: self
 class(atomicIonizationRateCollisionalClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (atomicIonizationRateCollisionalClass), intent(inout) :: self
 integer                                      , intent(in   ) :: stateFile
 type   (fgsl_file                           ), intent(in   ) :: fgslStateFile
 integer(c_size_t                            ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`atomicIonizationRateCollisionalVerner1996` Atomic collisional ionization rates are computed based on the code originally written by Dima Verner.

Atomic dielectronic recombination rates.

Additional implementations for atomic dielectronic recombination rates. are added using the `atomicRecombinationRateDiele` class. The implementation should be placed in a file containing the directive:

```
!# <atomicRecombinationRateDielectronic name="atomicRecombinationRateDielectronicMyImplementation">
!# <description>A short description of the implementation.</description>
!# </atomicRecombinationRateDielectronic>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `atomicRecombinationRateDielectronicMyImplementation`. The file *must* define a type that extends the `atomicRecombinationRateDielectronicClass` class (or extends another type which is itself an extension of the `atomicRecombinationRateDielectronicClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (atomicRecombinationRateDielectronicClass), intent(inout)          :: self
  logical                                          , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (atomicRecombinationRateDielectronicClass), intent(inout)           :: self
 type   (inputParameters                         ), intent(inout)           :: &
&  descriptor
 logical                                          , intent(in   ), optional :: &
&  includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(atomicRecombinationRateDielectronicClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (atomicRecombinationRateDielectronicClass), intent(inout) :: self
 integer                                          , intent(in   ) :: stateFile
 type   (fgsl_file                                ), intent(in   ) :: fgslStateFile
 integer(c_size_t                                 ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (atomicRecombinationRateDielectronicClass)                         , intent&
&  (inout) :: self
 type     (varying_string                          ), allocatable, dimension(:), intent&
&  (inout) :: allowedParameters
 character(len=*                                   )                         , intent&
&  (in   ) :: sourceName
end subroutine myImplementationAllowedParameters
```

`rate` Return the dielectroninc recombination rate (in units of $\mathrm{cm}^3\ \mathrm{s}^{-1}$) for the ion of given `atomicNumber` and `electronNumber` at the given `temperature` (in Kelvin). Must have the following interface:

```
double precision function myImplementationRate(self,atomicNumber, electronNumber,&
&  temperature)
 class           (atomicRecombinationRateDielectronicClass), intent(inout) :: self
 integer                                                   , intent(in   ) :: &
&  atomicNumber, electronNumber
 double precision                                          , intent(in   ) :: &
&  temperature
end double precision function myImplementationRate
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(atomicRecombinationRateDielectronicClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(atomicRecombinationRateDielectronicClass), intent(inout) :: self
 class(atomicRecombinationRateDielectronicClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (atomicRecombinationRateDielectronicClass), intent(inout) :: self
 integer                                          , intent(in   ) :: stateFile
 type   (fgsl_file                                ), intent(in   ) :: fgslStateFile
 integer(c_size_t                                 ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`atomicRecombinationRateDielectronicArnaud1985` Implements an atomic dielectronic recombination class which uses the fits from Aldrovandi and Pequignot [1973], Shull and van Steenberg [1982] and Arnaud and Rothenflug [1985].

Atomic Radiative Recombination

Additional implementations for atomic radiative recombination are added using the `atomicRecombinationRateRadiative` class. The implementation should be placed in a file containing the directive:

```
!# <atomicRecombinationRateRadiative name="atomicRecombinationRateRadiativeMyImplementation">
!# <description>A short description of the implementation.</description>
!# </atomicRecombinationRateRadiative>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `atomicRecombinationRateRadiativeMyImplementation`. The file *must* define a type that extends the `atomicRecombinationRateRadiativeClass` class (or extends another type which is itself an extension of the `atomicRecombinationRateRadiativeClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (atomicRecombinationRateRadiativeClass), intent(inout)          :: self
```

```
  logical                                        , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationDescriptor(self,descriptor,includeMethod)
  class  (atomicRecombinationRateRadiativeClass), intent(inout)           :: self
  type   (inputParameters                      ), intent(inout)           :: descriptor
  logical                                       , intent(in   ), optional :: &
&  includeMethod
 end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAutoHook(self)
  class(atomicRecombinationRateRadiativeClass), intent(inout) :: self
 end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
  class  (atomicRecombinationRateRadiativeClass), intent(inout) :: self
  integer                                       , intent(in   ) :: stateFile
  type   (fgsl_file                             ), intent(in   ) :: fgslStateFile
  integer(c_size_t                              ), intent(in   ) :: stateOperationID
 end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
  class    (atomicRecombinationRateRadiativeClass)                          , intent(&
&  inout) :: self
  type     (varying_string                       ), allocatable, dimension(:), intent(&
&  inout) :: allowedParameters
  character(len=*                                )                          , intent(in&
&     ) :: sourceName
 end subroutine myImplementationAllowedParameters
```

`rate` Returns the radiative recombination rate. Must have the following interface:

```
 double precision function myImplementationRate(self,atomicNumber, ionizationState,&
&  temperature,level)
  class           (atomicRecombinationRateRadiativeClass), intent(inout)           :: &
&  self
  integer                                                , intent(in   )           :: &
&  atomicNumber, ionizationState
  double precision                                       , intent(in   )           :: &
&  temperature
  integer                                                , intent(in   ), optional :: &
&  level
 end double precision function myImplementationRate
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(atomicRecombinationRateRadiativeClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(atomicRecombinationRateRadiativeClass), intent(inout) :: self
 class(atomicRecombinationRateRadiativeClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (atomicRecombinationRateRadiativeClass), intent(inout) :: self
 integer                                       , intent(in   ) :: stateFile
 type   (fgsl_file                            ), intent(in   ) :: fgslStateFile
 integer(c_size_t                             ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`atomicRecombinationRateRadiativeVerner1996` Atomic radiative recombination rates are computed based on the code originally written by Dima Verner.

Black Hole Binaries Initial Separation

Additional implementations for black hole binaries initial separation are added using the `blackHoleBinaryInitialSeparatio` class. The implementation should be placed in a file containing the directive:

```
!# <blackHoleBinaryInitialSeparation name="blackHoleBinaryInitialSeparationMyImplementation">
!# <description>A short description of the implementation.</description>
!# </blackHoleBinaryInitialSeparation>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `blackHoleBinaryInitialSeparationMyImplementation`. The file *must* define a type that extends the `blackHoleBinaryInitialSeparationClass` class (or extends another type which is itself an extension of the `blackHoleBinaryInitialSeparationClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`separationInitial` Computes the initial separation of a newly formed black hole binary black holes. Must have the following interface:

```
double precision function myImplementationSeparationInitial(self,node, nodeHost)
 class(blackHoleBinaryInitialSeparationClass), intent(inout)         :: self
 type (treeNode                              ), intent(inout), target :: node, nodeHost
end double precision function myImplementationSeparationInitial
```

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (blackHoleBinaryInitialSeparationClass), intent(inout)           :: self
  logical                                       , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationDescriptor(self,descriptor,includeMethod)
  class  (blackHoleBinaryInitialSeparationClass), intent(inout)           :: self
  type   (inputParameters                      ), intent(inout)           :: descriptor
  logical                                       , intent(in   ), optional :: &
&  includeMethod
 end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAutoHook(self)
  class(blackHoleBinaryInitialSeparationClass), intent(inout) :: self
 end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
  class  (blackHoleBinaryInitialSeparationClass), intent(inout) :: self
  integer                                       , intent(in   ) :: stateFile
  type   (fgsl_file                             ), intent(in   ) :: fgslStateFile
  integer(c_size_t                              ), intent(in   ) :: stateOperationID
 end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
  class    (blackHoleBinaryInitialSeparationClass)                          , intent(&
&  inout) :: self
  type     (varying_string                       ), allocatable, dimension(:), intent(&
&  inout) :: allowedParameters
  character(len=*                                )                          , intent(in&
&     ) :: sourceName
 end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationObjectType(self)
  class(blackHoleBinaryInitialSeparationClass), intent(inout) :: self
 end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(blackHoleBinaryInitialSeparationClass), intent(inout) :: self
 class(blackHoleBinaryInitialSeparationClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (blackHoleBinaryInitialSeparationClass), intent(inout) :: self
 integer                                       , intent(in   ) :: stateFile
 type   (fgsl_file                            ), intent(in   ) :: fgslStateFile
 integer(c_size_t                             ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`blackHoleBinaryInitialSeparationSpheroidRadiusFraction` A black hole binary initial separation class in which the radius is a fixed fraction of the scale radius of the larger of the host and satellite spheroids.

`blackHoleBinaryInitialSeparationTidalRadius` A black hole binary initial separation class in which the radius is based on tidal disruption of the satellite galaxy.

`blackHoleBinaryInitialSeparationVolonteri2003` A black hole binary initial separation class based on the model of Volonteri et al. [2003].

Black Hole Binaries Merger

Additional implementations for black hole binaries merger are added using the `blackHoleBinaryMerger` class. The implementation should be placed in a file containing the directive:

```
!# <blackHoleBinaryMerger name="blackHoleBinaryMergerMyImplementation">
!# <description>A short description of the implementation.</description>
!# </blackHoleBinaryMerger>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `blackHoleBinary-MergerMyImplementation`. The file *must* define a type that extends the `blackHoleBinaryMergerClass` class (or extends another type which is itself an extension of the `blackHoleBinaryMergerClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (blackHoleBinaryMergerClass), intent(inout)          :: self
  logical                            , intent(in   ), optional :: includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationDescriptor(self,descriptor,includeMethod)
  class  (blackHoleBinaryMergerClass), intent(inout)          :: self
  type   (inputParameters           ), intent(inout)          :: descriptor
  logical                            , intent(in   ), optional :: includeMethod
 end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAutoHook(self)
  class(blackHoleBinaryMergerClass), intent(inout) :: self
 end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
  class  (blackHoleBinaryMergerClass), intent(inout) :: self
  integer                            , intent(in   ) :: stateFile
  type   (fgsl_file                  ), intent(in   ) :: fgslStateFile
  integer(c_size_t                   ), intent(in   ) :: stateOperationID
 end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
  class    (blackHoleBinaryMergerClass)                          , intent(inout) :: &
&  self
  type     (varying_string            ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
  character(len=*                     )                          , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`merge` The the properties of the black hole resulting from a binary merger. Must have the following interface:

```
 subroutine myImplementationMerge(self,blackHoleMassA    , blackHoleMassB, &
&  blackHoleSpinA    , blackHoleSpinB,blackHoleMassFinal               , &
&  blackHoleSpinFinal)
  class           (blackHoleBinaryMergerClass), intent(inout) :: self
  double precision                            , intent(in   ) :: blackHoleMassA, &
&  blackHoleMassB, blackHoleSpinA, blackHoleSpinB
  double precision                            , intent(  out) :: blackHoleMassFinal, &
&  blackHoleSpinFinal
 end subroutine myImplementationMerge
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(blackHoleBinaryMergerClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(blackHoleBinaryMergerClass), intent(inout) :: self
 class(blackHoleBinaryMergerClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (blackHoleBinaryMergerClass), intent(inout) :: self
 integer                            , intent(in   ) :: stateFile
 type   (fgsl_file                 ), intent(in   ) :: fgslStateFile
 integer(c_size_t                  ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`blackHoleBinaryMergerRezzolla2008` A black hole binary merger class in which the black hole mass and spin resulting from binary mergers utilizing the approximations of Rezzolla et al. [2008].

Black Hole Binaries Recoil

Additional implementations for black hole binaries recoil are added using the `blackHoleBinaryRecoil` class. The implementation should be placed in a file containing the directive:

```
!# <blackHoleBinaryRecoil name="blackHoleBinaryRecoilMyImplementation">
!# <description>A short description of the implementation.</description>
!# </blackHoleBinaryRecoil>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `blackHoleBinaryRecoilMyImplementation`. The file *must* define a type that extends the `blackHoleBinaryRecoilClass` class (or extends another type which is itself an extension of the `blackHoleBinaryRecoilClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (blackHoleBinaryRecoilClass), intent(inout)          :: self
  logical                            , intent(in   ), optional :: includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (blackHoleBinaryRecoilClass), intent(inout)          :: self
 type   (inputParameters           ), intent(inout)          :: descriptor
 logical                            , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(blackHoleBinaryRecoilClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (blackHoleBinaryRecoilClass), intent(inout) :: self
 integer                            , intent(in   ) :: stateFile
 type   (fgsl_file                  ), intent(in   ) :: fgslStateFile
 integer(c_size_t                   ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (blackHoleBinaryRecoilClass)                          , intent(inout) :: &
&  self
 type     (varying_string            ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                     )                          , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`velocity` Computes the recoil velocity of the given pair of merging black holes. Must have the following interface:

```
double precision function myImplementationVelocity(self,blackHole1, blackHole2)
 class(blackHoleBinaryRecoilClass), intent(inout) :: self
 class(nodeComponentBlackHole    ), intent(inout) :: blackHole1, blackHole2
end double precision function myImplementationVelocity
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(blackHoleBinaryRecoilClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(blackHoleBinaryRecoilClass), intent(inout) :: self
 class(blackHoleBinaryRecoilClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (blackHoleBinaryRecoilClass), intent(inout) :: self
 integer                            , intent(in   ) :: stateFile
 type   (fgsl_file                 ), intent(in   ) :: fgslStateFile
 integer(c_size_t                  ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`blackHoleBinaryRecoilCampanelli2007` A black hole binary recoil class in which the recoil velocity is always campanelli2007.

`blackHoleBinaryRecoilZero` A black hole binary recoil class in which the recoil velocity is always zero.

Black Hole Binaries Separation Growth Rate

Additional implementations for black hole binaries separation growth rate are added using the `blackHoleBinarySeparationGr` class. The implementation should be placed in a file containing the directive:

```
!# <blackHoleBinarySeparationGrowthRate name="blackHoleBinarySeparationGrowthRateMyImplementation">
!# <description>A short description of the implementation.</description>
!# </blackHoleBinarySeparationGrowthRate>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `blackHoleBinarySeparationGrowthRateMyImplementation`. The file *must* define a type that extends the `blackHoleBinarySeparationGrowthRateClass` class (or extends another type which is itself an extension of the `blackHoleBinarySeparationGrowthRateClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (blackHoleBinarySeparationGrowthRateClass), intent(inout)          :: self
  logical                                          , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationDescriptor(self,descriptor,includeMethod)
  class  (blackHoleBinarySeparationGrowthRateClass), intent(inout)          :: self
  type   (inputParameters                         ), intent(inout)          :: &
&  descriptor
  logical                                          , intent(in   ), optional :: &
&  includeMethod
 end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAutoHook(self)
  class(blackHoleBinarySeparationGrowthRateClass), intent(inout) :: self
 end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
  class  (blackHoleBinarySeparationGrowthRateClass), intent(inout) :: self
  integer                                          , intent(in   ) :: stateFile
  type   (fgsl_file                                ), intent(in   ) :: fgslStateFile
  integer(c_size_t                                 ), intent(in   ) :: stateOperationID
 end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
  class    (blackHoleBinarySeparationGrowthRateClass)                         , intent&
&  (inout) :: self
  type     (varying_string                          ), allocatable, dimension(:), intent&
&  (inout) :: allowedParameters
  character(len=*                                   )                         , intent&
&  (in   ) :: sourceName
 end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationObjectType(self)
  class(blackHoleBinarySeparationGrowthRateClass), intent(inout) :: self
 end type(varying_string) function myImplementationObjectType
```

`growthRate` Computes the rate of growth of the separation of the given black hole and its binary companion in units of Mpc/Gyr. Must have the following interface:

```
double precision function myImplementationGrowthRate(self,blackHole)
 class(blackHoleBinarySeparationGrowthRateClass), intent(inout) :: self
 class(nodeComponentBlackHole                  ), intent(inout) :: blackHole
end double precision function myImplementationGrowthRate
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(blackHoleBinarySeparationGrowthRateClass), intent(inout) :: self
 class(blackHoleBinarySeparationGrowthRateClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (blackHoleBinarySeparationGrowthRateClass), intent(inout) :: self
 integer                                          , intent(in   ) :: stateFile
 type   (fgsl_file                                ), intent(in   ) :: fgslStateFile
 integer(c_size_t                                 ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`blackHoleBinarySeparationGrowthRateStandard` A black hole binary separation growth class which follows a modified version of Volonteri et al. [2003], including terms for dynamical friction, hardening due to scattering of stars and emission of gravitational waves.

`blackHoleBinarySeparationGrowthRateZero` A black hole binary separation growth class in which the separation does not grow.

Chemical Reaction Rates

Additional implementations for chemical reaction rates are added using the `chemicalReactionRate` class. The implementation should be placed in a file containing the directive:

```
!# <chemicalReactionRate name="chemicalReactionRateMyImplementation">
!# <description>A short description of the implementation.</description>
!# </chemicalReactionRate>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `chemicalReaction-RateMyImplementation`. The file *must* define a type that extends the `chemicalReactionRateClass` class (or extends another type which is itself an extension of the `chemicalReactionRateClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (chemicalReactionRateClass), intent(inout)           :: self
  logical                           , intent(in   ), optional :: includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationDescriptor(self,descriptor,includeMethod)
  class  (chemicalReactionRateClass), intent(inout)           :: self
  type   (inputParameters          ), intent(inout)           :: descriptor
  logical                           , intent(in   ), optional :: includeMethod
 end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAutoHook(self)
  class(chemicalReactionRateClass), intent(inout) :: self
 end subroutine myImplementationAutoHook
```

`rates` Return the collisional excitation cooling rate , in units of $\mathrm{J/m^3/s}$, for ion of given `atomicNumber` and `electronNumber` at temperature `T` (in Kelvin). Must have the following interface:

```
 subroutine myImplementationRates(self,temperature,chemicalDensity,radiation,&
&  chemicalRates,node)
  class           (chemicalReactionRateClass), intent(inout) :: self
  double precision                           , intent(in   ) :: temperature
  type            (chemicalAbundances       ), intent(in   ) :: chemicalDensity
  class           (radiationFieldClass      ), intent(inout) :: radiation
  type            (chemicalAbundances       ), intent(inout) :: chemicalRates
  type            (treeNode                 ), intent(inout) :: node
 end subroutine myImplementationRates
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
  class  (chemicalReactionRateClass), intent(inout) :: self
  integer                           , intent(in   ) :: stateFile
  type   (fgsl_file                ), intent(in   ) :: fgslStateFile
  integer(c_size_t                 ), intent(in   ) :: stateOperationID
 end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
  class    (chemicalReactionRateClass)                          , intent(inout) :: self
```

```
  type     (varying_string          ), allocatable, dimension(:), intent(inout) :: &
& allowedParameters
  character(len=*                   )                           , intent(in   ) :: &
& sourceName
 end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(chemicalReactionRateClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(chemicalReactionRateClass), intent(inout) :: self
 class(chemicalReactionRateClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (chemicalReactionRateClass), intent(inout) :: self
 integer                           , intent(in   ) :: stateFile
 type   (fgsl_file                ), intent(in   ) :: fgslStateFile
 integer(c_size_t                 ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`chemicalReactionRateZero` A chemical reaction rate class in which all rates are zero.

`chemicalReactionRateHydrogenNetwork` A chemical reaction rates for hydrogen using the fits from Abel et al. [1997] and Tegmark et al. [1997].

Chemical State

Additional implementations for chemical state are added using the `chemicalState` class. The implementation should be placed in a file containing the directive:

```
!# <chemicalState name="chemicalStateMyImplementation">
!# <description>A short description of the implementation.</description>
!# </chemicalState>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `chemicalStateMyImplementation`. The file *must* define a type that extends the `chemicalStateClass` class (or extends another type which is itself an extension of the `chemicalStateClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (chemicalStateClass), intent(inout)          :: self
  logical                    , intent(in   ), optional :: includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationDescriptor(self,descriptor,includeMethod)
  class  (chemicalStateClass), intent(inout)          :: self
  type   (inputParameters   ), intent(inout)          :: descriptor
  logical                    , intent(in   ), optional :: includeMethod
 end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAutoHook(self)
  class(chemicalStateClass), intent(inout) :: self
 end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
  class  (chemicalStateClass), intent(inout) :: self
  integer                    , intent(in   ) :: stateFile
  type   (fgsl_file         ), intent(in   ) :: fgslStateFile
  integer(c_size_t          ), intent(in   ) :: stateOperationID
 end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
  class    (chemicalStateClass)                          , intent(inout) :: self
  type     (varying_string    ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
  character(len=*             )                          , intent(in   ) :: sourceName
 end subroutine myImplementationAllowedParameters
```

`electronDensity` Return the electron density at the given temperature and hydrogen density for the specified set of abundances and radiation field. Units of the returned electron density are cm^-3. Must have the following interface:

```
 double precision function myImplementationElectronDensity(self,numberDensityHydrogen, &
&  temperature,gasAbundances,radiation)
  class           (chemicalStateClass ), intent(inout) :: self
  double precision                     , intent(in   ) :: numberDensityHydrogen, &
&  temperature
```

```
 type            (abundances         ), intent(in   ) :: gasAbundances
 class           (radiationFieldClass), intent(inout) :: radiation
end double precision function myImplementationElectronDensity
```

`electronDensityTemperatureLogSlope` Return the logarithmic gradient of electron density with temperature at the given temperature and hydrogen density for the specified set of abundances and radiation field. Must have the following interface:

```
double precision function myImplementationElectronDensityTemperatureLogSlope(self,&
&  numberDensityHydrogen, temperature,gasAbundances,radiation)
 class           (chemicalStateClass ), intent(inout) :: self
 double precision                     , intent(in   ) :: numberDensityHydrogen, &
&  temperature
 type            (abundances         ), intent(in   ) :: gasAbundances
 class           (radiationFieldClass), intent(inout) :: radiation
end double precision function myImplementationElectronDensityTemperatureLogSlope
```

`chemicalDensities` Return the densities of chemical species at the given temperature and hydrogen density for the specified set of abundances and radiation field. Units of the returned electron density are cm^{-}3. Must have the following interface:

```
subroutine myImplementationChemicalDensities(self,chemicalDensities,&
&  numberDensityHydrogen, temperature,gasAbundances,radiation)
 class           (chemicalStateClass ), intent(inout) :: self
 type            (chemicalAbundances ), intent(inout) :: chemicalDensities
 double precision                     , intent(in   ) :: numberDensityHydrogen, &
&  temperature
 type            (abundances         ), intent(in   ) :: gasAbundances
 class           (radiationFieldClass), intent(inout) :: radiation
end subroutine myImplementationChemicalDensities
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(chemicalStateClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(chemicalStateClass), intent(inout) :: self
 class(chemicalStateClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (chemicalStateClass), intent(inout) :: self
 integer                    , intent(in   ) :: stateFile
 type   (fgsl_file         ), intent(in   ) :: fgslStateFile
 integer(c_size_t          ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

`electronDensityDensityLogSlope` Return the logarithmic gradient of electron density with respect to density at the given temperature and hydrogen density for the specified set of abundances and radiation field. Must have the following interface:

```
 double precision function myImplementationElectronDensityDensityLogSlope(self,&
&  numberDensityHydrogen, temperature,gasAbundances,radiation)
  class           (chemicalStateClass ), intent(inout) :: self
  double precision                     , intent(in   ) :: numberDensityHydrogen, &
&  temperature
  type            (abundances         ), intent(in   ) :: gasAbundances
  class           (radiationFieldClass), intent(inout) :: radiation
 end double precision function myImplementationElectronDensityDensityLogSlope
```

Existing implementations are:

`chemicalStateCIEFile` Class providing chemical state via interpolation of tabulated values read from file. The HDF5 file containing the table should have the following form:

```
HDF5 "chemicalState.hdf5" {
GROUP "/" {
ATTRIBUTE "fileFormat" {
DATATYPE  H5T_STRING {
STRSIZE 1;
STRPAD H5T_STR_NULLTERM;
CSET H5T_CSET_ASCII;
CTYPE H5T_C_S1;
}
DATASPACE  SCALAR
DATA {
(0): "1"
}
}
DATASET "electronDensity" {
DATATYPE  H5T_IEEE_F64LE
DATASPACE  SIMPLE { ( 7, 8 ) / ( 7, 8 ) }
}
DATASET "hiDensity" {
DATATYPE  H5T_IEEE_F64LE
DATASPACE  SIMPLE { ( 7, 8 ) / ( 7, 8 ) }
}
DATASET "hiiDensity" {
DATATYPE  H5T_IEEE_F64LE
DATASPACE  SIMPLE { ( 7, 8 ) / ( 7, 8 ) }
}
DATASET "metallicity" {
DATATYPE  H5T_IEEE_F64LE
DATASPACE  SIMPLE { ( 8 ) / ( 8 ) }
ATTRIBUTE "extrapolateHigh" {
DATATYPE  H5T_STRING {
STRSIZE 3;
STRPAD H5T_STR_NULLTERM;
```

```
CSET H5T_CSET_ASCII;
CTYPE H5T_C_S1;
}
DATASPACE  SCALAR
DATA {
(0): "fix"
}
}
ATTRIBUTE "extrapolateLow" {
DATATYPE  H5T_STRING {
STRSIZE 3;
STRPAD H5T_STR_NULLTERM;
CSET H5T_CSET_ASCII;
CTYPE H5T_C_S1;
}
DATASPACE  SCALAR
DATA {
(0): "fix"
}
}
}
DATASET "temperature" {
DATATYPE  H5T_IEEE_F64LE
DATASPACE  SIMPLE { ( 7 ) / ( 7 ) }
ATTRIBUTE "extrapolateHigh" {
DATATYPE  H5T_STRING {
STRSIZE 3;
STRPAD H5T_STR_NULLTERM;
CSET H5T_CSET_ASCII;
CTYPE H5T_C_S1;
}
DATASPACE  SCALAR
DATA {
(0): "fix"
}
}
ATTRIBUTE "extrapolateLow" {
DATATYPE  H5T_STRING {
STRSIZE 3;
STRPAD H5T_STR_NULLTERM;
CSET H5T_CSET_ASCII;
CTYPE H5T_C_S1;
}
DATASPACE  SCALAR
DATA {
(0): "fix"
}
}
}
```

```
}
}
```

The `temperature` dataset should specify temperature (in Kelvin), while the `metallicity` dataset should give the logarithmic metallcity relative to Solar (a value of -999 or less is taken to imply zero metallicity). The `electronDensity` dataset should specify the number density of electrons relative to hydrogen at each temperature/metallicity pair. Optionally `hiDensity` and `hiiDensity` datasets may be added giving the number densities of HI and HII relative to hydrogen respectively The `extrapolateLow` and `extrapolateHigh` attributes of the `temperature` and `metallicity` datasets specify how the cooling rate should be extrapolated in the low and high vale limits. Allowed options for these attributes are:

`zero` The electron density is set to zero beyond the relevant limit.

`fixed` The electron density is held fixed at the value at the relevant limit.

`power law` The electron density is extrapolated assuming a power-law dependence beyond the relevant limit. This option is only allowed if the electron density is everywhere positive.

If the electron density is everywhere positive the interpolation will be done in the logarithmic of temperature, metallicity[3] and electron density. Otherwise, interpolation is linear in these quantities. The electron density is scaled assuming a linear dependence on hydrogen density.

`chemicalStateAtomicCIECloudy` Class providing chemical state by utilizing the CLOUDY code to compute state in collisional ionization equilibrium. CLOUDY will be downloaded, compiled and run automatically if necessary[4].

Infall rates in cold mode accretion.

Additional implementations for infall rates in cold mode accretion. are added using the `coldModeInfallRate` class. The implementation should be placed in a file containing the directive:

```
!# <coldModeInfallRate name="coldModeInfallRateMyImplementation">
!# <description>A short description of the implementation.</description>
!# </coldModeInfallRate>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `coldModeInfallRateMyImplementation`. The file *must* define a type that extends the `coldModeInfallRateClass` class (or extends another type which is itself an extension of the `coldModeInfallRateClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

[3] The exception is if the first electron density is tabulated for zero metallicity. In that case, a linear interpolation in metallicity is always used between zero and the first non-zero tabulated metallicity.

[4] CLOUDY is used to generate a file which contains a tabulation of the chemical state suitable for reading by the `CIE from file` method. Generation of the tabulation typically takes several hours, but only needs to be done once as the stored table is simply read back in on later runs.

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (coldModeInfallRateClass), intent(inout)           :: self
  logical                         , intent(in   ), optional :: includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (coldModeInfallRateClass), intent(inout)           :: self
 type   (inputParameters        ), intent(inout)           :: descriptor
 logical                         , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(coldModeInfallRateClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (coldModeInfallRateClass), intent(inout) :: self
 integer                         , intent(in   ) :: stateFile
 type   (fgsl_file               ), intent(in   ) :: fgslStateFile
 integer(c_size_t                ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`infallRate` Returns the cold mode infall rate for `thisNode` (in units of $M_\odot$ Gyr^{-1}). Must have the following interface:

```
double precision function myImplementationInfallRate(self,node)
 class(coldModeInfallRateClass), intent(inout) :: self
 type (treeNode                ), intent(inout) :: node
end double precision function myImplementationInfallRate
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (coldModeInfallRateClass)                          , intent(inout) :: self
 type     (varying_string         ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                  )                          , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(coldModeInfallRateClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(coldModeInfallRateClass), intent(inout) :: self
 class(coldModeInfallRateClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (coldModeInfallRateClass), intent(inout) :: self
 integer                         , intent(in   ) :: stateFile
 type   (fgsl_file              ), intent(in   ) :: fgslStateFile
 integer(c_size_t               ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`coldModeInfallRateDynamicalTime` A dynamicalTime cooling time calculation (based on the ratio of the thermal energy density to the volume cooling rate). See §12.7.1.

Conditional Mass Function

Additional implementations for conditional mass function are added using the `conditionalMassFunction` class. The implementation should be placed in a file containing the directive:

```
!# <conditionalMassFunction name="conditionalMassFunctionMyImplementation">
!# <description>A short description of the implementation.</description>
!# </conditionalMassFunction>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `conditionalMassFunctionMyImplementation`. The file *must* define a type that extends the `conditionalMassFunctionClass` class (or extends another type which is itself an extension of the `conditionalMassFunctionClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`massFunction` Return the cumulative conditional mass function, $\langle N(M_\star|M_{\rm halo})\rangle \equiv \phi(M_\star|M_{\rm halo})$. Must have the following interface:

```
 double precision function myImplementationMassFunction(self,massHalo  , mass,galaxyType&
&  )
  class           (conditionalMassFunctionClass), intent(inout)           :: self
  double precision                              , intent(in   )           :: massHalo, &
&  mass
  integer                                       , intent(in   ), optional :: galaxyType
 end double precision function myImplementationMassFunction
```

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (conditionalMassFunctionClass), intent(inout)           :: self
  logical                              , intent(in   ), optional :: includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationDescriptor(self,descriptor,includeMethod)
  class  (conditionalMassFunctionClass), intent(inout)           :: self
  type   (inputParameters             ), intent(inout)           :: descriptor
  logical                              , intent(in   ), optional :: includeMethod
 end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAutoHook(self)
  class(conditionalMassFunctionClass), intent(inout) :: self
 end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
  class  (conditionalMassFunctionClass), intent(inout) :: self
  integer                              , intent(in   ) :: stateFile
  type   (fgsl_file                   ), intent(in   ) :: fgslStateFile
  integer(c_size_t                    ), intent(in   ) :: stateOperationID
 end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
  class    (conditionalMassFunctionClass)                          , intent(inout) :: &
&  self
  type     (varying_string              ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
  character(len=*                       )                          , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationObjectType(self)
  class(conditionalMassFunctionClass), intent(inout) :: self
 end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(conditionalMassFunctionClass), intent(inout) :: self
 class(conditionalMassFunctionClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`massFunctionVariance` Return the variance in the cumulative conditional mass function, $\langle N(M_\star|M_{\rm halo})\rangle \equiv \phi(M_\star|M_{\rm halo})$. Must have the following interface:

```
double precision function myImplementationMassFunctionVariance(self,massHalo,massLow,&
&  massHigh)
 class           (conditionalMassFunctionClass), intent(inout) :: self
 double precision                              , intent(in   ) :: massHalo, massLow, &
&  massHigh
 end double precision function myImplementationMassFunctionVariance
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (conditionalMassFunctionClass), intent(inout) :: self
 integer                              , intent(in   ) :: stateFile
 type   (fgsl_file                    ), intent(in   ) :: fgslStateFile
 integer(c_size_t                     ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`conditionalMassFunctionBehroozi2010` A class which implements the conditional mass function using the fiting functions of Behroozi et al. [2010]. See §12.8.1.

Cooling Function

Additional implementations for cooling function are added using the `coolingFunction` class. The implementation should be placed in a file containing the directive:

```
!# <coolingFunction name="coolingFunctionMyImplementation">
!# <description>A short description of the implementation.</description>
!# </coolingFunction>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `coolingFunctionMyImplementation`. The file *must* define a type that extends the `coolingFunctionClass` class (or extends another type which is itself an extension of the `coolingFunctionClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (coolingFunctionClass), intent(inout)          :: self
  logical                      , intent(in   ), optional :: includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationDescriptor(self,descriptor,includeMethod)
  class  (coolingFunctionClass), intent(inout)          :: self
  type   (inputParameters     ), intent(inout)          :: descriptor
  logical                      , intent(in   ), optional :: includeMethod
 end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAutoHook(self)
  class(coolingFunctionClass), intent(inout) :: self
 end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
  class  (coolingFunctionClass), intent(inout) :: self
  integer                      , intent(in   ) :: stateFile
  type   (fgsl_file           ), intent(in   ) :: fgslStateFile
  integer(c_size_t            ), intent(in   ) :: stateOperationID
 end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
  class    (coolingFunctionClass)                          , intent(inout) :: self
  type     (varying_string      ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
  character(len=*               )                          , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`coolingFunction` Return the cooling function at the given temperature and hydrogen density for the specified set of abundances and radiation field. Units of the returned cooling function are the traditional ergs $\mathrm{cm}^{-}3\ \mathrm{s}^{-1}$. Must have the following interface:

```
 double precision function myImplementationCoolingFunction(self,numberDensityHydrogen, &
&  temperature,gasAbundances,chemicalDensities,radiation)
  class           (coolingFunctionClass), intent(inout) :: self
  double precision                      , intent(in   ) :: numberDensityHydrogen, &
&  temperature
  type            (abundances          ), intent(in   ) :: gasAbundances
  type            (chemicalAbundances  ), intent(in   ) :: chemicalDensities
  class           (radiationFieldClass ), intent(inout) :: radiation
 end double precision function myImplementationCoolingFunction
```

`coolingFunctionDensityLogSlope` Return $\mathrm{d}\ln\Lambda/\mathrm{d}\ln\rho$ for a cooling function at the given temperature and hydrogen density for the specified set of abundances and radiation field. Must have the following interface:

```
 double precision function myImplementationCoolingFunctionDensityLogSlope(self,&
&  numberDensityHydrogen, temperature,gasAbundances,chemicalDensities,radiation)
  class           (coolingFunctionClass), intent(inout) :: self
  double precision                      , intent(in   ) :: numberDensityHydrogen, &
&  temperature
  type            (abundances          ), intent(in   ) :: gasAbundances
  type            (chemicalAbundances  ), intent(in   ) :: chemicalDensities
  class           (radiationFieldClass ), intent(inout) :: radiation
 end double precision function myImplementationCoolingFunctionDensityLogSlope
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(coolingFunctionClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(coolingFunctionClass), intent(inout) :: self
 class(coolingFunctionClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (coolingFunctionClass), intent(inout) :: self
 integer                      , intent(in   ) :: stateFile
 type   (fgsl_file           ), intent(in   ) :: fgslStateFile
 integer(c_size_t            ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

`coolingFunctionTemperatureLogSlope` Return $\mathrm{d}\ln\Lambda/\mathrm{d}\ln T$ for a cooling function at the given temperature and hydrogen density for the specified set of abundances and radiation field. Must have the following interface:

```
 double precision function myImplementationCoolingFunctionTemperatureLogSlope(self,&
&  numberDensityHydrogen, temperature,gasAbundances,chemicalDensities,radiation)
  class           (coolingFunctionClass), intent(inout) :: self
  double precision                      , intent(in   ) :: numberDensityHydrogen, &
&  temperature
  type            (abundances          ), intent(in   ) :: gasAbundances
  type            (chemicalAbundances  ), intent(in   ) :: chemicalDensities
  class           (radiationFieldClass ), intent(inout) :: radiation
 end double precision function myImplementationCoolingFunctionTemperatureLogSlope
```

Existing implementations are:

`coolingFunctionCMBCompton` Class providing a cooling function due to Compton scattering off of CMB photons.

`coolingFunctionSummation` Class providing a cooling function which sums over other cooling functions.

`coolingFunctionMolecularHydrogenGalliPalla` Cooling function class which implements cooling from molecular hydrogen using the cooling function of Galli and Palla [1998].

`coolingFunctionAtomicCIECloudy` Class providing cooling function by utilizing the Cloudy code to compute cooling in collisional ionization equilibrium. Cloudy will be downloaded, compiled and run automatically if necessary[5].

`coolingFunctionCIEFile` Class providing a cooling function interpolated from a table read from file. The HDF5 file containing the table should have the following form:

```
HDF5 "coolingFunction.hdf5" {
GROUP "/" {
ATTRIBUTE "fileFormat" {
DATATYPE  H5T_STRING {
STRSIZE 1;
STRPAD H5T_STR_NULLTERM;
CSET H5T_CSET_ASCII;
CTYPE H5T_C_S1;
}
DATASPACE  SCALAR
DATA {
(0): "1"
}
}
DATASET "coolingRate" {
DATATYPE  H5T_IEEE_F64LE
DATASPACE  SIMPLE { ( 7, 8 ) / ( 7, 8 ) }
}
DATASET "metallicity" {
DATATYPE  H5T_IEEE_F64LE
DATASPACE  SIMPLE { ( 8 ) / ( 8 ) }
ATTRIBUTE "extrapolateHigh" {
DATATYPE  H5T_STRING {
STRSIZE 3;
STRPAD H5T_STR_NULLTERM;
CSET H5T_CSET_ASCII;
CTYPE H5T_C_S1;
}
DATASPACE  SCALAR
DATA {
(0): "fix"
}
}
```

[5] Cloudy is used to generate a file which contains a tabulation of the cooling function suitable for reading by the `CIE from file` method. Generation of the tabulation typically takes several hours, but only needs to be done once as the stored table is simply read back in on later runs.

```
ATTRIBUTE "extrapolateLow" {
DATATYPE  H5T_STRING {
STRSIZE 3;
STRPAD H5T_STR_NULLTERM;
CSET H5T_CSET_ASCII;
CTYPE H5T_C_S1;
}
DATASPACE  SCALAR
DATA {
(0): "fix"
}
}
}
DATASET "temperature" {
DATATYPE  H5T_IEEE_F64LE
DATASPACE  SIMPLE { ( 7 ) / ( 7 ) }
ATTRIBUTE "extrapolateHigh" {
DATATYPE  H5T_STRING {
STRSIZE 8;
STRPAD H5T_STR_NULLTERM;
CSET H5T_CSET_ASCII;
CTYPE H5T_C_S1;
}
DATASPACE  SCALAR
DATA {
(0): "powerLaw"
}
}
ATTRIBUTE "extrapolateLow" {
DATATYPE  H5T_STRING {
STRSIZE 8;
STRPAD H5T_STR_NULLTERM;
CSET H5T_CSET_ASCII;
CTYPE H5T_C_S1;
}
DATASPACE  SCALAR
DATA {
(0): "powerLaw"
}
}
}
}
}
```

The `temperature` dataset should specify temperature (in Kelvin), while the `metallicity` dataset should give the logarithmic metallcity relative to Solar (a value of -999 or less is taken to imply zero metallicity). The `coolingRate` dataset should specify the cooling function (in ergs cm^3 s^{-1} computed for a hydrogen density of 1 cm^{-3}) respectively at each temperature/metallicity pair. The `extrapolateLow` and `extrapolateHigh` attributes of the `temperature` and `metallicity` datasets specify how the cooling rate should be extrapolated in the low and high vale limits. Allowed options

for these attributes are:

`zero` The cooling function is set to zero beyond the relevant limit.

`fixed` The cooling function is held fixed at the value at the relevant limit.

`powerLaw` The cooling function is extrapolated assuming a power-law dependence beyond the relevant limit. This option is only allowed if the cooling function is everywhere positive.

If the cooling function is everywhere positive the interpolation will be done in the logarithm of temperature, metallicity[6] and cooling function. Otherwise, interpolation is linear in these quantities. The cooling function is scaled assuming a quadratic dependence on hydrogen density.

Cooling Infall Radius

Additional implementations for cooling infall radius are added using the `coolingInfallRadius` class. The implementation should be placed in a file containing the directive:

```
!# <coolingInfallRadius name="coolingInfallRadiusMyImplementation">
!# <description>A short description of the implementation.</description>
!# </coolingInfallRadius>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `coolingInfallRadiusMyImplementation`. The file *must* define a type that extends the `coolingInfallRadiusClass` class (or extends another type which is itself an extension of the `coolingInfallRadiusClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`radiusIncreaseRate` Return the rate at which the infall radius grows for `node` (in units of Mpc/Gyr). Must have the following interface:

```
double precision function myImplementationRadiusIncreaseRate(self,node)
 class(coolingInfallRadiusClass), intent(inout) :: self
 type (treeNode                ), intent(inout) :: node
end double precision function myImplementationRadiusIncreaseRate
```

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
 class  (coolingInfallRadiusClass), intent(inout)          :: self
 logical                          , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

[6] The exception is if the first cooling function is tabulated for zero metallicity. In that case, a linear interpolation in metallicity is always used between zero and the first non-zero tabulated metallicity.

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (coolingInfallRadiusClass), intent(inout)           :: self
 type   (inputParameters         ), intent(inout)           :: descriptor
 logical                          , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(coolingInfallRadiusClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`radius` Return the infall radius for `node` (in units of Mpc). Must have the following interface:

```
double precision function myImplementationRadius(self,node)
 class(coolingInfallRadiusClass), intent(inout) :: self
 type (treeNode                ), intent(inout) :: node
end double precision function myImplementationRadius
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (coolingInfallRadiusClass), intent(inout) :: self
 integer                          , intent(in   ) :: stateFile
 type   (fgsl_file                ), intent(in   ) :: fgslStateFile
 integer(c_size_t                 ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (coolingInfallRadiusClass)                          , intent(inout) :: self
 type     (varying_string          ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                   )                          , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(coolingInfallRadiusClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(coolingInfallRadiusClass), intent(inout) :: self
 class(coolingInfallRadiusClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (coolingInfallRadiusClass), intent(inout) :: self
 integer                          , intent(in   ) :: stateFile
 type   (fgsl_file               ), intent(in   ) :: fgslStateFile
 integer(c_size_t                ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`coolingInfallRadiusCoolingRadius` A simple infall radius calculation, simply assuming that the infall radius equals the cooling radius.

`coolingInfallRadiusCoolingFreefall` An infall radius calculation in which the infall radius is the smaller of the cooling and freefall radii.

Cooling radii.

Additional implementations for cooling radii. are added using the `coolingRadius` class. The implementation should be placed in a file containing the directive:

```
!# <coolingRadius name="coolingRadiusMyImplementation">
!# <description>A short description of the implementation.</description>
!# </coolingRadius>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `coolingRadiusMyImplementation`. The file *must* define a type that extends the `coolingRadiusClass` class (or extends another type which is itself an extension of the `coolingRadiusClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (coolingRadiusClass), intent(inout)           :: self
  logical                    , intent(in   ), optional :: includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (coolingRadiusClass), intent(inout)           :: self
 type   (inputParameters   ), intent(inout)           :: descriptor
 logical                    , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`radiusGrowthRate` Returns the rate of increase of the cooling radius for gas in the hot atmosphere surrounding the galaxy in `node` in units of Mpc/Gyr. Must have the following interface:

```
double precision function myImplementationRadiusGrowthRate(self,node)
 class(coolingRadiusClass), intent(inout) :: self
 type (treeNode          ), intent(inout) :: node
end double precision function myImplementationRadiusGrowthRate
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(coolingRadiusClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`radius` Returns the cooling radius for gas in the hot atmosphere surrounding the galaxy in `node` in units of Mpc. Must have the following interface:

```
double precision function myImplementationRadius(self,node)
 class(coolingRadiusClass), intent(inout)         :: self
 type (treeNode          ), intent(inout), target :: node
end double precision function myImplementationRadius
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (coolingRadiusClass), intent(inout) :: self
 integer                    , intent(in   ) :: stateFile
 type   (fgsl_file         ), intent(in   ) :: fgslStateFile
 integer(c_size_t          ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (coolingRadiusClass)                            , intent(inout) :: self
 type     (varying_string    ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*             )                            , intent(in   ) :: sourceName
end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(coolingRadiusClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(coolingRadiusClass), intent(inout) :: self
 class(coolingRadiusClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (coolingRadiusClass), intent(inout) :: self
 integer                    , intent(in   ) :: stateFile
 type   (fgsl_file         ), intent(in   ) :: fgslStateFile
 integer(c_size_t          ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`coolingRadiusIsothermal` A cooling radius class for isothermal halos. Computes the cooling radius by assuming that the hot gas density profile is an isothermal profile ($\rho(r) \propto r^{-2}$), and that the cooling rate scales as density squared, $\dot{E} \propto \rho^2$, such that the cooling time scales as inverse density, $t_{\rm cool} \propto \rho^{-1}$. Consequently, the cooling radius grows as the square root of the time available for cooling.

`coolingRadiusBetaProfile` A cooling radius class for β-profile halos. Computes the cooling radius by assuming that the hot gas density profile is a β-profile ($\rho(r) \propto [r^2+r_{\rm c}^2]^{-1}$), and that the cooling rate scales as density squared, $\dot{E} \propto \rho^2$, such that the cooling time scales as inverse density, $t_{\rm cool} \propto \rho^{-1}$. Consequently, the cooling radius is given by

$$r_{\rm cool} = r_{\rm virial} \left(\left[\frac{t_{\rm avail}}{t_0} - 1 \right] \left[\frac{t_{\rm virial}}{t_0} - 1 \right]^{-1} \right)^{1/2}, \tag{16.2}$$

where t_0, and $t_{\rm virial}$ are the cooling times at zero radius and the virial radius respectively.

`coolingRadiusSimple` A cooling radius class computes the cooling radius by seeking the radius at which the time available for cooling equals the cooling time. The growth rate is determined consistently based on the slope of the density profile, the density dependence of the cooling function and the rate at which the time available for cooling is increasing. This method assumes that the cooling time is a monotonic function of radius.

Cooling rates.

Additional implementations for cooling rates. are added using the `coolingRate` class. The implementation should be placed in a file containing the directive:

```
!# <coolingRate name="coolingRateMyImplementation">
!# <description>A short description of the implementation.</description>
!# </coolingRate>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `coolingRateMyImplementation`. The file *must* define a type that extends the `coolingRateClass` class (or

extends another type which is itself an extension of the `coolingRateClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
& )
 class  (coolingRateClass), intent(inout)          :: self
 logical                  , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (coolingRateClass), intent(inout)          :: self
 type   (inputParameters ), intent(inout)          :: descriptor
 logical                  , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(coolingRateClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (coolingRateClass), intent(inout) :: self
 integer                  , intent(in   ) :: stateFile
 type   (fgsl_file       ), intent(in   ) :: fgslStateFile
 integer(c_size_t        ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (coolingRateClass)                          , intent(inout) :: self
 type     (varying_string  ), allocatable, dimension(:), intent(inout) :: &
& allowedParameters
 character(len=*           )                          , intent(in   ) :: sourceName
end subroutine myImplementationAllowedParameters
```

`rate` Returns the cooling rate of gas in the hot atmosphere surrounding the galaxy in `node` in units of $M_\odot$/Gyr. Must have the following interface:

```
double precision function myImplementationRate(self,node)
 class(coolingRateClass), intent(inout) :: self
 type (treeNode        ), intent(inout) :: node
end double precision function myImplementationRate
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(coolingRateClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(coolingRateClass), intent(inout) :: self
 class(coolingRateClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (coolingRateClass), intent(inout) :: self
 integer                  , intent(in   ) :: stateFile
 type   (fgsl_file       ), intent(in   ) :: fgslStateFile
 integer(c_size_t        ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`coolingRateSimpleScaling` A cooling rate class in which the cooling rate scales with the mass of the halo.

`coolingRateZero` A cooling rate class in which the cooling rate is always zero.

`coolingRateMultiplier` A cooling rate class which modifies another cooling rate by multiplying the rate by a fixed value.

`coolingRateSimple` A cooling rate class in which the cooling rate equals the mass of hot gas divided by a fixed timescale.

`coolingRateCutOff` A cooling rate class which modifies another cooling rate by cutting off cooling above some virial velocity.

`coolingRateCole2000` Computes the mass cooling rate in a hot gas halo utilizing the Cole et al. [2000] method. This is based on the properties of the halo at formation time, and gives a zero cooling rate when the cooling radius exceeds the virial radius.

`coolingRateVelocityMaximumScaling` A cooling rate class in which the cooling rate scales with the peak circular velocity in the halo.

`coolingRateWhiteFrenk1991` A cooling rate class for the White and Frenk [1991] cooling rate calculation.

`coolingRateNoCoolingSatellites` A cooling rate class which modifies another cooling rate by cutting off cooling in satellites

Specific angular momentua of cooling gas.

Additional implementations for specific angular momentua of cooling gas. are added using the `coolingSpecificAngularMomen` class. The implementation should be placed in a file containing the directive:

```
!# <coolingSpecificAngularMomentum name="coolingSpecificAngularMomentumMyImplementation">
!# <description>A short description of the implementation.</description>
!# </coolingSpecificAngularMomentum>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `coolingSpecificAngularMomentumMyImplementation`. The file *must* define a type that extends the `coolingSpecificAngularMomentumClass` class (or extends another type which is itself an extension of the `coolingSpecificAngularMomentumClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (coolingSpecificAngularMomentumClass), intent(inout)           :: self
  logical                                     , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (coolingSpecificAngularMomentumClass), intent(inout)           :: self
 type   (inputParameters                    ), intent(inout)           :: descriptor
 logical                                     , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`angularMomentumSpecific` Return the specific angular momentum (in units of km/s Mpc) of cooling gas in `node`. Must have the following interface:

```
double precision function myImplementationAngularMomentumSpecific(self,node,radius)
 class           (coolingSpecificAngularMomentumClass), intent(inout) :: self
 type            (treeNode                           ), intent(inout) :: node
 double precision                                     , intent(in   ) :: radius
end double precision function myImplementationAngularMomentumSpecific
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(coolingSpecificAngularMomentumClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (coolingSpecificAngularMomentumClass), intent(inout) :: self
 integer                                     , intent(in   ) :: stateFile
 type   (fgsl_file                           ), intent(in   ) :: fgslStateFile
 integer(c_size_t                            ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (coolingSpecificAngularMomentumClass)                          , intent(&
&  inout) :: self
 type     (varying_string                     ), allocatable, dimension(:), intent(&
&  inout) :: allowedParameters
 character(len=*                              )                          , intent(in  &
&   ) :: sourceName
 end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(coolingSpecificAngularMomentumClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(coolingSpecificAngularMomentumClass), intent(inout) :: self
 class(coolingSpecificAngularMomentumClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (coolingSpecificAngularMomentumClass), intent(inout) :: self
 integer                                     , intent(in   ) :: stateFile
 type   (fgsl_file                           ), intent(in   ) :: fgslStateFile
 integer(c_size_t                            ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`coolingSpecificAngularMomentumMean` A specific angular momentum of cooling gas class in which all gas has the mean specific angular momentum of the hot gas halo.

`coolingSpecificAngularMomentumConstantRotation` A specific angular momentum of cooling gas class which assumes a constant rotation velocity as a function of radius.

Cooling times.

Additional implementations for cooling times. are added using the `coolingTime` class. The implementation should be placed in a file containing the directive:

```
!# <coolingTime name="coolingTimeMyImplementation">
!# <description>A short description of the implementation.</description>
!# </coolingTime>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `coolingTimeMyImplementation`. The file *must* define a type that extends the `coolingTimeClass` class (or extends another type which is itself an extension of the `coolingTimeClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
 class  (coolingTimeClass), intent(inout)           :: self
 logical                  , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (coolingTimeClass), intent(inout)           :: self
 type   (inputParameters ), intent(inout)           :: descriptor
 logical                  , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(coolingTimeClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`time` Returns the cooling time for gas in the hot atmosphere surrounding the galaxy in units of Gyr. Must have the following interface:

```
double precision function myImplementationTime(self,temperature      , density,&
&  gasAbundances,chemicalDensities,radiation)
 class           (coolingTimeClass   ), intent(inout) :: self
 double precision                     , intent(in   ) :: temperature, density
 type            (abundances         ), intent(in   ) :: gasAbundances
 type            (chemicalAbundances ), intent(in   ) :: chemicalDensities
 class           (radiationFieldClass), intent(inout) :: radiation
end double precision function myImplementationTime
```

`gradientTemperatureLogarithmic` Returns the logarithmic derivative of cooling time with respect to temperature for gas in the hot atmosphere surrounding the galaxy. Must have the following interface:

```
double precision function myImplementationGradientTemperatureLogarithmic(self,&
&  temperature      , density,gasAbundances,chemicalDensities,radiation)
 class           (coolingTimeClass   ), intent(inout) :: self
 double precision                     , intent(in   ) :: temperature, density
 type            (abundances         ), intent(in   ) :: gasAbundances
 type            (chemicalAbundances ), intent(in   ) :: chemicalDensities
 class           (radiationFieldClass), intent(inout) :: radiation
end double precision function myImplementationGradientTemperatureLogarithmic
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (coolingTimeClass), intent(inout) :: self
 integer                  , intent(in   ) :: stateFile
 type   (fgsl_file       ), intent(in   ) :: fgslStateFile
 integer(c_size_t        ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`gradientDensityLogarithmic` Returns the logarithmic derivative of cooling time with respect to density for gas in the hot atmosphere surrounding the galaxy. Must have the following interface:

```
double precision function myImplementationGradientDensityLogarithmic(self,temperature  &
&      , density,gasAbundances,chemicalDensities,radiation)
 class           (coolingTimeClass   ), intent(inout) :: self
 double precision                     , intent(in   ) :: temperature, density
 type            (abundances         ), intent(in   ) :: gasAbundances
 type            (chemicalAbundances ), intent(in   ) :: chemicalDensities
 class           (radiationFieldClass), intent(inout) :: radiation
end double precision function myImplementationGradientDensityLogarithmic
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (coolingTimeClass)                          , intent(inout) :: self
 type     (varying_string  ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*           )                          , intent(in   ) :: sourceName
end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(coolingTimeClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(coolingTimeClass), intent(inout) :: self
 class(coolingTimeClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (coolingTimeClass), intent(inout) :: self
 integer                  , intent(in   ) :: stateFile
 type   (fgsl_file       ), intent(in   ) :: fgslStateFile
 integer(c_size_t        ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`coolingTimeSimple` A simple cooling time calculation (based on the ratio of the thermal energy density to the volume cooling rate). See §12.9.8.

Time available for cooling

Additional implementations for time available for cooling are added using the `coolingTimeAvailable` class. The implementation should be placed in a file containing the directive:

```
!# <coolingTimeAvailable name="coolingTimeAvailableMyImplementation">
!# <description>A short description of the implementation.</description>
!# </coolingTimeAvailable>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `coolingTimeAvailable-MyImplementation`. The file *must* define a type that extends the `coolingTimeAvailableClass` class (or extends another type which is itself an extension of the `coolingTimeAvailableClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
 class  (coolingTimeAvailableClass), intent(inout)          :: self
 logical                           , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (coolingTimeAvailableClass), intent(inout)          :: self
 type   (inputParameters          ), intent(inout)          :: descriptor
 logical                           , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(coolingTimeAvailableClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (coolingTimeAvailableClass), intent(inout) :: self
 integer                           , intent(in   ) :: stateFile
 type   (fgsl_file                ), intent(in   ) :: fgslStateFile
 integer(c_size_t                 ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (coolingTimeAvailableClass)                          , intent(inout) :: self
 type     (varying_string           ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                    )                          , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`timeAvailable` Return the time available for cooling in `node` in units of Gyr. Must have the following interface:

```
double precision function myImplementationTimeAvailable(self,node)
 class(coolingTimeAvailableClass), intent(inout) :: self
 type (treeNode                 ), intent(inout) :: node
end double precision function myImplementationTimeAvailable
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(coolingTimeAvailableClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(coolingTimeAvailableClass), intent(inout) :: self
 class(coolingTimeAvailableClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`timeAvailableIncreaseRate` Return the rate at which the time available for cooling increases in `node` (dimensionless). Must have the following interface:

```
double precision function myImplementationTimeAvailableIncreaseRate(self,node)
 class(coolingTimeAvailableClass), intent(inout) :: self
 type (treeNode                 ), intent(inout) :: node
end double precision function myImplementationTimeAvailableIncreaseRate
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (coolingTimeAvailableClass), intent(inout) :: self
 integer                           , intent(in   ) :: stateFile
 type   (fgsl_file                 ), intent(in   ) :: fgslStateFile
 integer(c_size_t                  ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`coolingTimeAvailableFormationTime` A time available for cooling class which implements the algorithm of Cole et al. [2000], that is, the time available is the time since the halo "formed".

`coolingTimeAvailableWhiteFrenk1991` A time available for cooling class which implements the algorithm of White and Frenk [1991]. The time available is set to a value between the age of the Universe and the dynamical time of the halo, depending on the interpolating parameter `[ageFactor]`.

Mass Variance of Cosmological Density Field

Additional implementations for mass variance of cosmological density field are added using the `cosmologicalMassVariance` class. The implementation should be placed in a file containing the directive:

```
!# <cosmologicalMassVariance name="cosmologicalMassVarianceMyImplementation">
!# <description>A short description of the implementation.</description>
!# </cosmologicalMassVariance>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `cosmologicalMassVarianceMyImplementation`. The file *must* define a type that extends the `cosmologicalMassVarianceClass` class (or extends another type which is itself an extension of the `cosmologicalMassVarianceClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`powerNormalization` Return the normalization of the power spectrum. Must have the following interface:

```
double precision function myImplementationPowerNormalization(self)
 class(cosmologicalMassVarianceClass), intent(inout) :: self
end double precision function myImplementationPowerNormalization
```

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (cosmologicalMassVarianceClass), intent(inout)          :: self
  logical                               , intent(in   ), optional :: includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationDescriptor(self,descriptor,includeMethod)
  class  (cosmologicalMassVarianceClass), intent(inout)          :: self
  type   (inputParameters              ), intent(inout)          :: descriptor
  logical                               , intent(in   ), optional :: includeMethod
 end subroutine myImplementationDescriptor
```

`rootVarianceAndLogarithmicGradient` Return the value and logarithmic gradient with respect to mass of the root-variance of the cosmological density field. Must have the following interface:

```
 subroutine myImplementationRootVarianceAndLogarithmicGradient(self,mass,rootVariance, &
&  rootVarianceLogarithmicGradient)
  class           (cosmologicalMassVarianceClass), intent(inout) :: self
  double precision                               , intent(in   ) :: mass
  double precision                               , intent(  out) :: rootVariance, &
&  rootVarianceLogarithmicGradient
 end subroutine myImplementationRootVarianceAndLogarithmicGradient
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAutoHook(self)
  class(cosmologicalMassVarianceClass), intent(inout) :: self
 end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
  class  (cosmologicalMassVarianceClass), intent(inout) :: self
  integer                               , intent(in   ) :: stateFile
  type   (fgsl_file                    ), intent(in   ) :: fgslStateFile
  integer(c_size_t                     ), intent(in   ) :: stateOperationID
 end subroutine myImplementationStateStore
```

`sigma8` Return the value of σ_8. Must have the following interface:

```
 double precision function myImplementationSigma8(self)
  class(cosmologicalMassVarianceClass), intent(inout) :: self
 end double precision function myImplementationSigma8
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
  class    (cosmologicalMassVarianceClass)                          , intent(inout) :: &
&  self
  type     (varying_string              ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
  character(len=*                       )                          , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(cosmologicalMassVarianceClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`rootVarianceLogarithmicGradient` Return the logarithmic gradient of the root-variance of the cosmological density field with respect to mass. Must have the following interface:

```
double precision function myImplementationRootVarianceLogarithmicGradient(self,mass)
 class           (cosmologicalMassVarianceClass), intent(inout) :: self
 double precision                               , intent(in   ) :: mass
end double precision function myImplementationRootVarianceLogarithmicGradient
```

`mass` Return the mass corresponding to the given `rootVariance` of the cosmological density field. Must have the following interface:

```
double precision function myImplementationMass(self,rootVariance)
 class           (cosmologicalMassVarianceClass), intent(inout) :: self
 double precision                               , intent(in   ) :: rootVariance
end double precision function myImplementationMass
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(cosmologicalMassVarianceClass), intent(inout) :: self
 class(cosmologicalMassVarianceClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`rootVariance` Return the root-variance of the cosmological density field. Must have the following interface:

```
double precision function myImplementationRootVariance(self,mass)
 class           (cosmologicalMassVarianceClass), intent(inout) :: self
 double precision                               , intent(in   ) :: mass
end double precision function myImplementationRootVariance
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (cosmologicalMassVarianceClass), intent(inout) :: self
 integer                               , intent(in   ) :: stateFile
 type   (fgsl_file                    ), intent(in   ) :: fgslStateFile
 integer(c_size_t                     ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`cosmologicalMassVarianceFilteredPower` Mass variance of cosmological density fields computed from a filtered power spectrum. The normalization of the mass variance is specified via the `[sigma_8]` parameter, which defines the linear theory root-variance of the density field in spheres of radii $8h^{-1}$Mpc. The mass variance, $\sigma(M)$, is found by integration over the linear theory power spectrum, with the specified power spectrum window function. The fractional tolerance for this integration can be set via the `[tolerance]` parameter. (The normalization of $\sigma(M)$ to give the desired σ_8 always uses a top-hat window function. For this integration the tolerance can be set via the `[toleranceTopHat]` parameter.) This is tabulated across the required range. Cubic spline interpolation is then used to interpolate in this table to give $\sigma(M)$ at any required value of M. The tabulation is always forced to be monotonically decreasing with M. However, the interpolation is not necessarily monotonic—for example in cases where $\sigma(M)$ becomes constant or close to constant as a function of M the interpolation can become non-monotonic over some ranges of M. If strict monotonicity is required set `[monotonicInterpolation]`=`true`. This causes a monotonic spline interpolator to be used instead which gaurantees monotonicity.

`cosmologicalMassVariancePeakBackgroundSplit` The cosmological mass variance is computed by taking the variance from some other mass variance class, $\sigma^2(M)$, and offsetting it by the variance of the background in the peak-background split model, $\sigma^2(M_\mathrm{e})$, where M_e is the mass contained within the region defined as the background.

Cosmology Functions

Additional implementations for cosmology functions are added using the `cosmologyFunctions` class. The implementation should be placed in a file containing the directive:

```
!# <cosmologyFunctions name="cosmologyFunctionsMyImplementation">
!# <description>A short description of the implementation.</description>
!# </cosmologyFunctions>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `cosmologyFunctionsMyImplementation`. The file *must* define a type that extends the `cosmologyFunctionsClass` class (or extends another type which is itself an extension of the `cosmologyFunctionsClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hubbleParameterEpochal` Returns the Hubble parameter at the requested cosmological time, `time`, or expansion factor, `expansionFactor`. Must have the following interface:

```
 double precision function myImplementationHubbleParameterEpochal(self,time,&
&  expansionFactor,collapsingPhase)
  class           (cosmologyFunctionsClass), intent(inout)           :: self
  double precision                         , intent(in   ), optional :: time, &
&  expansionFactor
  logical                                  , intent(in   ), optional :: collapsingPhase
 end double precision function myImplementationHubbleParameterEpochal
```

`omegaDarkEnergyEpochal` Return the dark energy density parameter at expansion factor `expansionFactor`. Must have the following interface:

```
 double precision function myImplementationOmegaDarkEnergyEpochal(self,time  , &
&  expansionFactor,collapsingPhase)
  class           (cosmologyFunctionsClass), intent(inout)           :: self
  double precision                         , intent(in   ), optional :: time, &
&  expansionFactor
  logical                                  , intent(in   ), optional :: collapsingPhase
 end double precision function myImplementationOmegaDarkEnergyEpochal
```

`equationOfStateDarkEnergy` HASH(0x1fe6c28) Must have the following interface:

```
 double precision function myImplementationEquationOfStateDarkEnergy(self,time, &
&  expansionFactor)
  class           (cosmologyFunctionsClass), intent(inout)           :: self
  double precision                         , intent(in   ), optional :: time, &
&  expansionFactor
 end double precision function myImplementationEquationOfStateDarkEnergy
```

`expansionFactor` Returns the expansion factor at cosmological time `time`. Must have the following interface:

```
 double precision function myImplementationExpansionFactor(self,time)
  class           (cosmologyFunctionsClass), intent(inout) :: self
  double precision                         , intent(in   ) :: time
 end double precision function myImplementationExpansionFactor
```

`equalityEpochMatterRadiation` Return the epoch of matter-radiation magnitude equality (either expansion factor or cosmic time). Must have the following interface:

```
 double precision function myImplementationEqualityEpochMatterRadiation(self,requestType&
&  )
  class  (cosmologyFunctionsClass), intent(inout)           :: self
  integer                         , intent(in   ), optional :: requestType
 end double precision function myImplementationEqualityEpochMatterRadiation
```

`cosmicTime` Return the cosmological age at the given expansion factor. Must have the following interface:

```
 double precision function myImplementationCosmicTime(self,expansionFactor,&
&  collapsingPhase)
  class           (cosmologyFunctionsClass), intent(inout)           :: self
  double precision                         , intent(in   )           :: expansionFactor
  logical                                  , intent(in   ), optional :: collapsingPhase
 end double precision function myImplementationCosmicTime
```

`equalityEpochMatterDarkEnergy` Return the epoch of matter-dark energy magnitude equality (either expansion factor or cosmic time). Must have the following interface:

```
 double precision function myImplementationEqualityEpochMatterDarkEnergy(self,&
&  requestType)
  class  (cosmologyFunctionsClass), intent(inout)           :: self
  integer                         , intent(in   ), optional :: requestType
 end double precision function myImplementationEqualityEpochMatterDarkEnergy
```

`distanceComovingConvert` Convert between different measures of comoving distance. Must have the following interface:

```
 double precision function myImplementationDistanceComovingConvert(self,output,&
&  distanceLuminosity, distanceModulus, distanceModulusKCorrected, redshift)
  class           (cosmologyFunctionsClass), intent(inout)           :: self
  integer                                  , intent(in   )           :: output
  double precision                         , intent(in   ), optional :: &
&  distanceLuminosity, distanceModulus, distanceModulusKCorrected, redshift
 end double precision function myImplementationDistanceComovingConvert
```

`temperatureCMBEpochal` Return the temperature of the cosmic microwave background at `expansionFactor`. Must have the following interface:

```
 double precision function myImplementationTemperatureCMBEpochal(self,time  , &
&  expansionFactor,collapsingPhase)
  class           (cosmologyFunctionsClass), intent(inout)           :: self
  double precision                         , intent(in   ), optional :: time, &
&  expansionFactor
  logical                                  , intent(in   ), optional :: collapsingPhase
 end double precision function myImplementationTemperatureCMBEpochal
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationDescriptor(self,descriptor,includeMethod)
  class  (cosmologyFunctionsClass), intent(inout)           :: self
  type   (inputParameters        ), intent(inout)           :: descriptor
  logical                         , intent(in   ), optional :: includeMethod
 end subroutine myImplementationDescriptor
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
  class  (cosmologyFunctionsClass), intent(inout) :: self
  integer                         , intent(in   ) :: stateFile
  type   (fgsl_file               ), intent(in   ) :: fgslStateFile
  integer(c_size_t                ), intent(in   ) :: stateOperationID
 end subroutine myImplementationStateStore
```

`matterDensityEpochal` Convenience function that returns the matter density at the specified epoch. Must have the following interface:

```
 double precision function myImplementationMatterDensityEpochal(self,time           , &
&  expansionFactor,collapsingPhase)
  class           (cosmologyFunctionsClass), intent(inout)           :: self
  double precision                         , intent(in   ), optional :: time, &
&  expansionFactor
  logical                                  , intent(in   ), optional :: collapsingPhase
 end double precision function myImplementationMatterDensityEpochal
```

`hubbleParameterRateOfChange` Returns the rate of change of the Hubble parameter at the requested cosmological time, `time`, or expansion factor, `expansionFactor`. Must have the following interface:

```
 double precision function myImplementationHubbleParameterRateOfChange(self,time,&
&  expansionFactor,collapsingPhase)
  class           (cosmologyFunctionsClass), intent(inout)           :: self
  double precision                         , intent(in   ), optional :: time, &
&  expansionFactor
  logical                                  , intent(in   ), optional :: collapsingPhase
 end double precision function myImplementationHubbleParameterRateOfChange
```

`distanceLuminosity` Return the luminosity distance to the given cosmic `time`. Must have the following interface:

```
 double precision function myImplementationDistanceLuminosity(self,time)
  class           (cosmologyFunctionsClass), intent(inout) :: self
  double precision                         , intent(in   ) :: time
 end double precision function myImplementationDistanceLuminosity
```

`dominationEpochMatter` Compute the epoch at which matter dominates over other forms of energy by a given factor. Must have the following interface:

```
 double precision function myImplementationDominationEpochMatter(self,dominateFactor)
  class           (cosmologyFunctionsClass), intent(inout) :: self
  double precision                         , intent(in   ) :: dominateFactor
 end double precision function myImplementationDominationEpochMatter
```

`distanceComoving` Return the comoving distance to the given cosmic `time`. Must have the following interface:

```
 double precision function myImplementationDistanceComoving(self,time)
  class           (cosmologyFunctionsClass), intent(inout) :: self
  double precision                         , intent(in   ) :: time
 end double precision function myImplementationDistanceComoving
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationDeepCopy(self,destination)
  class(cosmologyFunctionsClass), intent(inout) :: self
  class(cosmologyFunctionsClass), intent(inout) :: destination
 end subroutine myImplementationDeepCopy
```

`epochTime` Convenience function that returns the time corresponding to an epoch specified by time or expansion factor. A default implementation exists. If overridden the following interface must be used:

```
 double precision function myImplementationEpochTime(self,time           , &
&  expansionFactor,collapsingPhase)
  class           (cosmologyFunctionsClass), intent(inout)           :: self
  double precision                         , intent(in   ), optional :: time, &
&  expansionFactor
  logical                                  , intent(in   ), optional :: collapsingPhase
 end double precision function myImplementationEpochTime
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (cosmologyFunctionsClass), intent(inout) :: self
 integer                         , intent(in   ) :: stateFile
 type   (fgsl_file              ), intent(in   ) :: fgslStateFile
 integer(c_size_t               ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

`timeAtDistanceComoving` Return the cosmic time corresponding to the given `comovingDistance`. Must have the following interface:

```
double precision function myImplementationTimeAtDistanceComoving(self,comovingDistance)
 class           (cosmologyFunctionsClass), intent(inout) :: self
 double precision                         , intent(in   ) :: comovingDistance
end double precision function myImplementationTimeAtDistanceComoving
```

`equalityEpochMatterCurvature` Return the epoch of matter-curvature magnitude equality (either expansion factor or cosmic time). Must have the following interface:

```
double precision function myImplementationEqualityEpochMatterCurvature(self,requestType&
&  )
 class  (cosmologyFunctionsClass), intent(inout)           :: self
 integer                         , intent(in   ), optional :: requestType
end double precision function myImplementationEqualityEpochMatterCurvature
```

`expansionRate` Returns the cosmological expansion rate, $\dot{a}/a$ at expansion factor `expansionFactor`. Must have the following interface:

```
double precision function myImplementationExpansionRate(self,expansionFactor)
 class           (cosmologyFunctionsClass), intent(inout) :: self
 double precision                         , intent(in   ) :: expansionFactor
end double precision function myImplementationExpansionRate
```

`expansionFactorFromRedshift` Returns expansion factor given a redshift. A default implementation exists. If overridden the following interface must be used:

```
double precision function myImplementationExpansionFactorFromRedshift(self,redshift)
 class           (cosmologyFunctionsClass), intent(inout) :: self
 double precision                         , intent(in   ) :: redshift
end double precision function myImplementationExpansionFactorFromRedshift
```

`timeBigCrunch` Return the cosmological age at Big Crunch (or a negative value if no Big Crunch occurs). Must have the following interface:

```
double precision function myImplementationTimeBigCrunch(self)
 class(cosmologyFunctionsClass), intent(inout) :: self
end double precision function myImplementationTimeBigCrunch
```

`distanceAngular` Return the angular diameter distance to the given cosmic `time`. Must have the following interface:

```
double precision function myImplementationDistanceAngular(self,time)
 class           (cosmologyFunctionsClass), intent(inout) :: self
 double precision                         , intent(in   ) :: time
end double precision function myImplementationDistanceAngular
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(cosmologyFunctionsClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`densityScalingEarlyTime` Compute the scaling of density with expansion factor at early times in the universe. Must have the following interface:

```
subroutine myImplementationDensityScalingEarlyTime(self,dominateFactor,densityPower  , &
&  expansionFactorDominant,OmegaDominant)
 class           (cosmologyFunctionsClass), intent(inout)           :: self
 double precision                         , intent(in   )           :: dominateFactor
 double precision                         , intent(  out)           :: densityPower, &
&  expansionFactorDominant
 double precision                         , intent(  out), optional :: OmegaDominant
end subroutine myImplementationDensityScalingEarlyTime
```

`omegaMatterEpochal` Return the matter density parameter at expansion factor `expansionFactor`. Must have the following interface:

```
double precision function myImplementationOmegaMatterEpochal(self,time  , &
&  expansionFactor,collapsingPhase)
 class           (cosmologyFunctionsClass), intent(inout)           :: self
 double precision                         , intent(in   ), optional :: time, &
&  expansionFactor
 logical                                  , intent(in   ), optional :: collapsingPhase
end double precision function myImplementationOmegaMatterEpochal
```

`comovingVolumeElementRedshift` Returns the differential comoving volume element $\mathrm{d}V/\mathrm{d}z = r_\mathrm{c}^2(t)\mathrm{c}H^{-1}(t)$ (where r_c is the comoving distance to time t and $H(t)$ is the Hubble parameter at that time) for unit solid angle at the specified `time`. A default implementation exists. If overridden the following interface must be used:

```
double precision function myImplementationComovingVolumeElementRedshift(self,time)
 class           (cosmologyFunctionsClass), intent(inout) :: self
 double precision                         , intent(in   ) :: time
end double precision function myImplementationComovingVolumeElementRedshift
```

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
 class  (cosmologyFunctionsClass), intent(inout)           :: self
 logical                         , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`exponentDarkEnergy` HASH(0x1fea6a0) Must have the following interface:

```
double precision function myImplementationExponentDarkEnergy(self,time, expansionFactor&
&  )
 class           (cosmologyFunctionsClass), intent(inout)           :: self
```

```
  double precision                          , intent(in   ), optional :: time, &
&  expansionFactor
 end double precision function myImplementationExponentDarkEnergy
```

`omegaMatterRateOfChange` Return the rate of change of the matter density parameter at expansion factor `expansionFactor`. Must have the following interface:

```
 double precision function myImplementationOmegaMatterRateOfChange(self,time  , &
&  expansionFactor,collapsingPhase)
  class           (cosmologyFunctionsClass), intent(inout)           :: self
  double precision                          , intent(in   ), optional :: time, &
&  expansionFactor
  logical                                   , intent(in   ), optional :: collapsingPhase
 end double precision function myImplementationOmegaMatterRateOfChange
```

`epochValidate` Check the given cosmic epoch is valid (aborting otherwise) and, optionally, return time or expansion factor associated with the epoch. Must have the following interface:

```
 subroutine myImplementationEpochValidate(self,timeIn,expansionFactorIn,collapsingIn,&
&  timeOut,expansionFactorOut,collapsingOut)
  class           (cosmologyFunctionsClass), intent(inout)           :: self
  double precision                          , intent(in   ), optional :: timeIn
  double precision                          , intent(in   ), optional :: &
&  expansionFactorIn
  logical                                   , intent(in   ), optional :: collapsingIn
  double precision                          , intent(  out), optional :: timeOut
  double precision                          , intent(  out), optional :: &
&  expansionFactorOut
  logical                                   , intent(  out), optional :: collapsingOut
 end subroutine myImplementationEpochValidate
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAutoHook(self)
  class(cosmologyFunctionsClass), intent(inout) :: self
 end subroutine myImplementationAutoHook
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
  class    (cosmologyFunctionsClass)                          , intent(inout) :: self
  type     (varying_string         ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
  character(len=*                  )                          , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`redshiftFromExpansionFactor` Returns redshift for a given expansion factor. A default implementation exists. If overridden the following interface must be used:

```
 double precision function myImplementationRedshiftFromExpansionFactor(self,&
&  expansionFactor)
```

```
 class           (cosmologyFunctionsClass), intent(inout) :: self
 double precision                         , intent(in   ) :: expansionFactor
end double precision function myImplementationRedshiftFromExpansionFactor
```

`comovingVolumeElementTime` Returns the differential comoving volume element $\mathrm{d}V/\mathrm{d}t = r_\mathrm{c}^2(t)ca(t)$ (where r_c is the comoving distance to time t and $a(t)$ is the expansion at that time) for unit solid angle at the specified `time`. A default implementation exists. If overridden the following interface must be used:

```
double precision function myImplementationComovingVolumeElementTime(self,time)
 class           (cosmologyFunctionsClass), intent(inout) :: self
 double precision                         , intent(in   ) :: time
end double precision function myImplementationComovingVolumeElementTime
```

Existing implementations are:

`cosmologyFunctionsMatterLambda` Cosmological relations are computed assuming a universe that contains only matter and a cosmological constant. See §12.4.1.

`cosmologyFunctionsMatterDarkEnergy` Cosmological relations are computed assuming a universe that contains only matter and dark energy with an equation of state $w(a) = w_0 + w_1 a(1-a)$. See §12.4.2.

`cosmologyFunctionsStaticUniverse` A cosmology functions class for static universes. Intended for testing purposes. Time is arbitrary (as there is no Big Bang), and expansion factor is fixed at 1. Attempts to compute time from expansion factor will cause fatal errors. Expansion rates and the Hubble constant are set to zero.

Cosmological Parameters

Additional implementations for cosmological parameters are added using the `cosmologyParameters` class. The implementation should be placed in a file containing the directive:

```
!# <cosmologyParameters name="cosmologyParametersMyImplementation">
!# <description>A short description of the implementation.</description>
!# </cosmologyParameters>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `cosmologyParameters-MyImplementation`. The file *must* define a type that extends the `cosmologyParametersClass` class (or extends another type which is itself an extension of the `cosmologyParametersClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`HubbleConstant` Return the Hubble constant at the present day. The optional `units` argument specifies if the return value should be in units of km/s/Mpc (hubbleUnitsStandard), Gyr^{-1} (hubbleUnitsTime), or 100 km/s/Mpc (hubbleUnitsLittleH). Must have the following interface:

```
double precision function myImplementationHubbleConstant(self,units)
 class  (cosmologyParametersClass), intent(inout)           :: self
 integer                          , intent(in   ), optional :: units
end double precision function myImplementationHubbleConstant
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(cosmologyParametersClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`temperatureCMB` Return the temperature of the cosmic microwave background radiation (in units of Kelvin) at the present day. Must have the following interface:

```
double precision function myImplementationTemperatureCMB(self)
 class(cosmologyParametersClass), intent(inout) :: self
end double precision function myImplementationTemperatureCMB
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (cosmologyParametersClass), intent(inout)           :: self
 type   (inputParameters         ), intent(inout)           :: descriptor
 logical                          , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
 class  (cosmologyParametersClass), intent(inout)           :: self
 logical                          , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`OmegaCurvature` Return the cosmological curvature density in units of the critical density at the present day. Must have the following interface:

```
double precision function myImplementationOmegaCurvature(self)
 class(cosmologyParametersClass), intent(inout) :: self
end double precision function myImplementationOmegaCurvature
```

`OmegaBaryon` Return the cosmological baryon density in units of the critical density at the present day. Must have the following interface:

```
double precision function myImplementationOmegaBaryon(self)
 class(cosmologyParametersClass), intent(inout) :: self
end double precision function myImplementationOmegaBaryon
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(cosmologyParametersClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`densityCritical` Return the critical density at the present day in units of $M_\odot/\mathrm{Mpc}^3$. Must have the following interface:

```
double precision function myImplementationDensityCritical(self)
 class(cosmologyParametersClass), intent(inout) :: self
end double precision function myImplementationDensityCritical
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (cosmologyParametersClass), intent(inout) :: self
 integer                          , intent(in   ) :: stateFile
 type   (fgsl_file               ), intent(in   ) :: fgslStateFile
 integer(c_size_t                ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`OmegaDarkEnergy` Return the cosmological dark energy density in units of the critical density at the present day. Must have the following interface:

```
double precision function myImplementationOmegaDarkEnergy(self)
 class(cosmologyParametersClass), intent(inout) :: self
end double precision function myImplementationOmegaDarkEnergy
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (cosmologyParametersClass)                          , intent(inout) :: self
 type     (varying_string          ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                   )                          , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`OmegaRadiation` Return the cosmological radiation density in units of the critical density at the present day. Must have the following interface:

```
double precision function myImplementationOmegaRadiation(self)
 class(cosmologyParametersClass), intent(inout) :: self
end double precision function myImplementationOmegaRadiation
```

`OmegaMatter` Return the cosmological matter density in units of the critical density at the present day. Must have the following interface:

```
double precision function myImplementationOmegaMatter(self)
 class(cosmologyParametersClass), intent(inout) :: self
end double precision function myImplementationOmegaMatter
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(cosmologyParametersClass), intent(inout) :: self
 class(cosmologyParametersClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (cosmologyParametersClass), intent(inout) :: self
 integer                          , intent(in   ) :: stateFile
 type   (fgsl_file               ), intent(in   ) :: fgslStateFile
 integer(c_size_t                ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`cosmologyParametersSimple` Provides basic cosmological parameters: $(H_0, \Omega_\mathrm{M}, \Omega_\Lambda, \Omega_\mathrm{b}, T_\mathrm{CMB})$. Also provides derived quantities $(\Omega_\mathrm{K}, \Omega_\mathrm{r}, \rho_\mathrm{crit})$.

Critical Overdensity

Additional implementations for critical overdensity are added using the `criticalOverdensity` class. The implementation should be placed in a file containing the directive:

```
!# <criticalOverdensity name="criticalOverdensityMyImplementation">
!# <description>A short description of the implementation.</description>
!# </criticalOverdensity>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `criticalOverdensity-MyImplementation`. The file *must* define a type that extends the `criticalOverdensityClass` class (or extends another type which is itself an extension of the `criticalOverdensityClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`timeOfCollapse` Returns the time of collapse for a perturbation of linear theory overdensity `criticalOverdensity`. A default implementation exists. If overridden the following interface must be used:

```
 double precision function myImplementationTimeOfCollapse(self,criticalOverdensity,mass,&
&  node)
  class           (criticalOverdensityClass), intent(inout)                  :: self
  double precision                          , intent(in   )                  :: &
&  criticalOverdensity
  double precision                          , intent(in   ), optional        :: mass
  type            (treeNode                ), intent(inout), optional, target :: node
 end double precision function myImplementationTimeOfCollapse
```

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (criticalOverdensityClass), intent(inout)          :: self
  logical                          , intent(in   ), optional :: includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (criticalOverdensityClass), intent(inout)          :: self
 type   (inputParameters         ), intent(inout)          :: descriptor
 logical                          , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(criticalOverdensityClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`value` Return the critical overdensity at the given time and mass. Must have the following interface:

```
 double precision function myImplementationValue(self,time      , expansionFactor,&
&  collapsing,mass,node)
  class           (criticalOverdensityClass), intent(inout)           :: self
  double precision                          , intent(in   ), optional :: time, &
&  expansionFactor
  logical                                   , intent(in   ), optional :: collapsing
  double precision                          , intent(in   ), optional :: mass
  type            (treeNode                ), intent(inout), optional :: node
 end double precision function myImplementationValue
```

`isMassDependent` Return true if the critical overdensity is dependent on the mass of the halo. Must have the following interface:

```
logical function myImplementationIsMassDependent(self)
 class(criticalOverdensityClass), intent(inout) :: self
end logical function myImplementationIsMassDependent
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (criticalOverdensityClass), intent(inout) :: self
 integer                          , intent(in   ) :: stateFile
 type   (fgsl_file                ), intent(in   ) :: fgslStateFile
 integer(c_size_t                 ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`collapsingMass` Return the mass scale just collapsing at the given cosmic time. A default implementation exists. If overridden the following interface must be used:

```
 double precision function myImplementationCollapsingMass(self,time      , &
&  expansionFactor,collapsing,node)
  class           (criticalOverdensityClass), intent(inout)                  :: self
  double precision                          , intent(in   ), optional        :: time, &
&  expansionFactor
  logical                                   , intent(in   ), optional        :: &
&  collapsing
  type            (treeNode                ), intent(inout), optional, target :: node
 end double precision function myImplementationCollapsingMass
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (criticalOverdensityClass)                          , intent(inout) :: self
 type     (varying_string          ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                   )                          , intent(in   ) :: &
&  sourceName
end subroutine myImplementationAllowedParameters
```

`gradientTime` Return the derivative with respect to time of the linear theory critical overdensity for collapse at the given cosmic time. Must have the following interface:

```
double precision function myImplementationGradientTime(self,time      , expansionFactor&
&  ,collapsing,mass,node)
 class           (criticalOverdensityClass), intent(inout)           :: self
 double precision                          , intent(in   ), optional :: time, &
&  expansionFactor
 logical                                   , intent(in   ), optional :: collapsing
 double precision                          , intent(in   ), optional :: mass
 type            (treeNode                ), intent(inout), optional :: node
end double precision function myImplementationGradientTime
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(criticalOverdensityClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(criticalOverdensityClass), intent(inout) :: self
 class(criticalOverdensityClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (criticalOverdensityClass), intent(inout) :: self
 integer                          , intent(in   ) :: stateFile
 type   (fgsl_file               ), intent(in   ) :: fgslStateFile
 integer(c_size_t                ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

`gradientMass` Return the derivative with respect to mass of the linear theory critical overdensity for collapse at the given cosmic time. Must have the following interface:

```
double precision function myImplementationGradientMass(self,time      , expansionFactor&
&  ,collapsing,mass,node)
 class           (criticalOverdensityClass), intent(inout)           :: self
```

```
      double precision                            , intent(in   ), optional :: time, &
     &  expansionFactor
      logical                                     , intent(in   ), optional :: collapsing
      double precision                            , intent(in   ), optional :: mass
      type            (treeNode                  ), intent(inout), optional :: node
     end double precision function myImplementationGradientMass
```

Existing implementations are:

`criticalOverdensityEnvironmental` The critical overdensity is given by some other critical overdensity class multiplied some environment-dependent factor.

`criticalOverdensitySphericalCollapseMatterLambda` Critical overdensity for collapse based on the spherical collapse in a matter plus cosmological constant universe (see, for example, Percival 2005).

`criticalOverdensitySphericalCollapseMatterDE` Critical overdensity for collapse based on the spherical collapse in a matter plus dark energy universe.

`criticalOverdensityKitayamaSuto1996` Provides a critical overdensity class based on the fitting functions of Kitayama and Suto [1996], and is therefore valid only for flat cosmological models.

`criticalOverdensityBarkana2001WDM` Provides a critical overdensity for collapse based on the warm dark matter (WDM) modifier of Barkana et al. [2001] applied to some other critical overdensity class.

`criticalOverdensityFixed` The critical overdensity is set to a fixed number divided by the linear growth factor.

`criticalOverdensityPeakBackgroundSplit` The ciritical overdensity is given by some other critical overdensity class offset by the halo envinronmental overdensity.

Dark matter halo biases.

Additional implementations for dark matter halo biases. are added using the `darkMatterHaloBias` class. The implementation should be placed in a file containing the directive:

```
!# <darkMatterHaloBias name="darkMatterHaloBiasMyImplementation">
!# <description>A short description of the implementation.</description>
!# </darkMatterHaloBias>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `darkMatterHaloBiasMyImplementation`. The file *must* define a type that extends the `darkMatterHaloBiasClass` class (or extends another type which is itself an extension of the `darkMatterHaloBiasClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (darkMatterHaloBiasClass), intent(inout)           :: self
  logical                         , intent(in   ), optional :: includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (darkMatterHaloBiasClass), intent(inout)           :: self
 type   (inputParameters        ), intent(inout)           :: descriptor
 logical                         , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(darkMatterHaloBiasClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (darkMatterHaloBiasClass), intent(inout) :: self
 integer                         , intent(in   ) :: stateFile
 type   (fgsl_file               ), intent(in   ) :: fgslStateFile
 integer(c_size_t                ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (darkMatterHaloBiasClass)                          , intent(inout) :: self
 type     (varying_string         ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                  )                          , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`biasByMass` Returns the bias of a halo specified by a mass (in $M_\odot$) and time (in Gyr). Must have the following interface:

```
double precision function myImplementationBiasByMass(self,mass, time)
 class           (darkMatterHaloBiasClass), intent(inout) :: self
 double precision                         , intent(in   ) :: mass, time
end double precision function myImplementationBiasByMass
```

`biasByNode` Returns the bias of the halo in the supplied node. A default implementation exists. If overridden the following interface must be used:

```
double precision function myImplementationBiasByNode(self,node)
 class(darkMatterHaloBiasClass), intent(inout) :: self
 type (treeNode               ), intent(inout) :: node
end double precision function myImplementationBiasByNode
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(darkMatterHaloBiasClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(darkMatterHaloBiasClass), intent(inout) :: self
 class(darkMatterHaloBiasClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (darkMatterHaloBiasClass), intent(inout) :: self
 integer                         , intent(in   ) :: stateFile
 type   (fgsl_file              ), intent(in   ) :: fgslStateFile
 integer(c_size_t               ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`darkMatterHaloBiasPressSchechter` A dark matter halo mass bias class utilizing the Press-Schechter algorithm [Mo and White, 1996].

`darkMatterHaloBiasSheth2001` A dark matter halo mass bias class utilizing the algorithm of Sheth et al. [2001].

`darkMatterHaloBiasTinker2010` A dark matter halo mass bias class utilizing the algorithm of Tinker et al. [2010].

Dark Matter Halo Mass Accretion Histories

Additional implementations for dark matter halo mass accretion histories are added using the `darkMatterHaloMassAccretionH` class. The implementation should be placed in a file containing the directive:

```
!# <darkMatterHaloMassAccretionHistory name="darkMatterHaloMassAccretionHistoryMyImplementation">
!# <description>A short description of the implementation.</description>
!# </darkMatterHaloMassAccretionHistory>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain

all functions required by the implementation after that line. Function names should begin with `darkMatterHaloMassAccretionHistoryMyImplementation`. The file *must* define a type that extends the `darkMatterHaloMassAccretionHistoryClass` class (or extends another type which is itself an extension of the `darkMatterHaloMassAccretionHistoryClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (darkMatterHaloMassAccretionHistoryClass), intent(inout)           :: self
  logical                                         , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationDescriptor(self,descriptor,includeMethod)
  class  (darkMatterHaloMassAccretionHistoryClass), intent(inout)           :: self
  type   (inputParameters                         ), intent(inout)           :: &
&  descriptor
  logical                                          , intent(in   ), optional :: &
&  includeMethod
 end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAutoHook(self)
  class(darkMatterHaloMassAccretionHistoryClass), intent(inout) :: self
 end subroutine myImplementationAutoHook
```

`time` Returns the time at which the given halo mass was reached. Must have the following interface:

```
 double precision function myImplementationTime(self,node,mass)
  class           (darkMatterHaloMassAccretionHistoryClass), intent(inout) :: self
  type            (treeNode                               ), intent(inout) :: node
  double precision                                         , intent(in   ) :: mass
 end double precision function myImplementationTime
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
  class  (darkMatterHaloMassAccretionHistoryClass), intent(inout) :: self
  integer                                         , intent(in   ) :: stateFile
  type   (fgsl_file                               ), intent(in   ) :: fgslStateFile
  integer(c_size_t                                ), intent(in   ) :: stateOperationID
 end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (darkMatterHaloMassAccretionHistoryClass)                          , intent(&
&  inout) :: self
 type     (varying_string                        ), allocatable, dimension(:), intent(&
&  inout) :: allowedParameters
 character(len=*                                 )                          , intent(&
&  in   ) :: sourceName
end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(darkMatterHaloMassAccretionHistoryClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(darkMatterHaloMassAccretionHistoryClass), intent(inout) :: self
 class(darkMatterHaloMassAccretionHistoryClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (darkMatterHaloMassAccretionHistoryClass), intent(inout) :: self
 integer                                         , intent(in   ) :: stateFile
 type   (fgsl_file                               ), intent(in   ) :: fgslStateFile
 integer(c_size_t                                ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`darkMatterHaloMassAccretionHistoryWechsler2002` Dark matter halo mass accretion histories using the Wechsler et al. [2002] algorithm.

`darkMatterHaloMassAccretionHistoryCorrea2015` Dark matter halo mass accretion histories using the Correa et al. [2015] algorithm.

`darkMatterHaloMassAccretionHistoryZhao2009` Dark matter halo mass accretion histories using the Zhao et al. [2009] algorithm.

Dark matter halo mass loss rates.

Additional implementations for dark matter halo mass loss rates. are added using the `darkMatterHaloMassLossRate` class. The implementation should be placed in a file containing the directive:

```
!# <darkMatterHaloMassLossRate name="darkMatterHaloMassLossRateMyImplementation">
!# <description>A short description of the implementation.</description>
!# </darkMatterHaloMassLossRate>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `darkMatterHaloMassLossRateMyImplementation`. The file *must* define a type that extends the `darkMatterHaloMassLossRateClass` class (or extends another type which is itself an extension of the `darkMatterHaloMassLossRateClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (darkMatterHaloMassLossRateClass), intent(inout)           :: self
  logical                                 , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (darkMatterHaloMassLossRateClass), intent(inout)           :: self
 type   (inputParameters                ), intent(inout)           :: descriptor
 logical                                 , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(darkMatterHaloMassLossRateClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (darkMatterHaloMassLossRateClass), intent(inout) :: self
 integer                                 , intent(in   ) :: stateFile
 type   (fgsl_file                       ), intent(in   ) :: fgslStateFile
 integer(c_size_t                        ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
  class    (darkMatterHaloMassLossRateClass)                          , intent(inout) &
&  :: self
  type     (varying_string                 ), allocatable, dimension(:), intent(inout) &
&  :: allowedParameters
```

```
 character(len=*                          )                          , intent(in    ) &
&  :: sourceName
 end subroutine myImplementationAllowedParameters
```

`rate` Returns the rate of mass loss (in $M_\odot$/Gyr) from `node`. Must have the following interface:

```
double precision function myImplementationRate(self,node)
 class(darkMatterHaloMassLossRateClass), intent(inout) :: self
 type (treeNode                       ), intent(inout) :: node
end double precision function myImplementationRate
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(darkMatterHaloMassLossRateClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(darkMatterHaloMassLossRateClass), intent(inout) :: self
 class(darkMatterHaloMassLossRateClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (darkMatterHaloMassLossRateClass), intent(inout) :: self
 integer                                 , intent(in   ) :: stateFile
 type   (fgsl_file                       ), intent(in   ) :: fgslStateFile
 integer(c_size_t                        ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`darkMatterHaloMassLossRateZero` A dark matter halo mass loss rate class which assumes a zero rate of mass loss.

`darkMatterHaloMassLossRateVanDenBosch` A dark matter halo mass loss rate class which uses the prescription of van den Bosch et al. [2005].

Dark Matter Halo Scales

Additional implementations for dark matter halo scales are added using the `darkMatterHaloScale` class. The implementation should be placed in a file containing the directive:

```
!# <darkMatterHaloScale name="darkMatterHaloScaleMyImplementation">
!# <description>A short description of the implementation.</description>
!# </darkMatterHaloScale>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `darkMatterHaloScaleMyImplementation`. The file *must* define a type that extends the `darkMatterHaloScaleClass` class (or extends another type which is itself an extension of the `darkMatterHaloScaleClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`virialVelocity` The virial velocity of a dark matter halo Must have the following interface:

```
double precision function myImplementationVirialVelocity(self,node)
 class(darkMatterHaloScaleClass), intent(inout) :: self
 type (treeNode                ), intent(inout) :: node
end double precision function myImplementationVirialVelocity
```

`meanDensityGrowthRate` The growth rate of the mean density of a dark matter halo. Must have the following interface:

```
double precision function myImplementationMeanDensityGrowthRate(self,node)
 class(darkMatterHaloScaleClass), intent(inout) :: self
 type (treeNode                ), intent(inout) :: node
end double precision function myImplementationMeanDensityGrowthRate
```

`virialTemperature` The virial temperature of a dark matter halo Must have the following interface:

```
double precision function myImplementationVirialTemperature(self,node)
 class(darkMatterHaloScaleClass), intent(inout) :: self
 type (treeNode                ), intent(inout) :: node
end double precision function myImplementationVirialTemperature
```

`dynamicalTimescale` The characteristic dynamical timescale of a dark matter halo. Must have the following interface:

```
double precision function myImplementationDynamicalTimescale(self,node)
 class(darkMatterHaloScaleClass), intent(inout) :: self
 type (treeNode                ), intent(inout) :: node
end double precision function myImplementationDynamicalTimescale
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(darkMatterHaloScaleClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`meanDensity` The mean density of a dark matter halo. Must have the following interface:

```
double precision function myImplementationMeanDensity(self,node)
 class(darkMatterHaloScaleClass), intent(inout) :: self
 type (treeNode                ), intent(inout) :: node
end double precision function myImplementationMeanDensity
```

`virialRadiusGrowthRate` The growth rate of the virial radius of a dark matter halo. Must have the following interface:

```
double precision function myImplementationVirialRadiusGrowthRate(self,node)
 class(darkMatterHaloScaleClass), intent(inout) :: self
 type (treeNode                ), intent(inout) :: node
end double precision function myImplementationVirialRadiusGrowthRate
```

`virialVelocityGrowthRate` The growth rate of the virial velocity of a dark matter halo. Must have the following interface:

```
double precision function myImplementationVirialVelocityGrowthRate(self,node)
 class(darkMatterHaloScaleClass), intent(inout) :: self
 type (treeNode                ), intent(inout) :: node
end double precision function myImplementationVirialVelocityGrowthRate
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (darkMatterHaloScaleClass), intent(inout)           :: self
 type   (inputParameters         ), intent(inout)           :: descriptor
 logical                          , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
 class  (darkMatterHaloScaleClass), intent(inout)           :: self
 logical                          , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(darkMatterHaloScaleClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`virialRadius` The virial radius of a dark matter halo. Must have the following interface:

```
double precision function myImplementationVirialRadius(self,node)
 class(darkMatterHaloScaleClass), intent(inout) :: self
 type (treeNode                ), intent(inout) :: node
end double precision function myImplementationVirialRadius
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (darkMatterHaloScaleClass), intent(inout) :: self
 integer                          , intent(in   ) :: stateFile
 type   (fgsl_file              ), intent(in   ) :: fgslStateFile
 integer(c_size_t               ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (darkMatterHaloScaleClass)                          , intent(inout) :: self
 type     (varying_string          ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                   )                          , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(darkMatterHaloScaleClass), intent(inout) :: self
 class(darkMatterHaloScaleClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (darkMatterHaloScaleClass), intent(inout) :: self
 integer                          , intent(in   ) :: stateFile
 type   (fgsl_file              ), intent(in   ) :: fgslStateFile
 integer(c_size_t               ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

`virialRadiusGradientLogarithmicMass` The logarithmic gradient of virial radius of a dark matter halo with halo mass at fixed epoch. Must have the following interface:

```
double precision function myImplementationVirialRadiusGradientLogarithmicMass(self,node&
&  )
 class(darkMatterHaloScaleClass), intent(inout) :: self
 type (treeNode                ), intent(inout) :: node
end double precision function myImplementationVirialRadiusGradientLogarithmicMass
```

Existing implementations are:

`darkMatterHaloScaleVirialDensityContrastDefinition` Dark matter halo scales derived from virial density contrasts.

Dark Matter Particle

Additional implementations for dark matter particle are added using the `darkMatterParticle` class. The implementation should be placed in a file containing the directive:

```
!# <darkMatterParticle name="darkMatterParticleMyImplementation">
!# <description>A short description of the implementation.</description>
!# </darkMatterParticle>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `darkMatterParticleMyImplementation`. The file *must* define a type that extends the `darkMatterParticleClass` class (or extends another type which is itself an extension of the `darkMatterParticleClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
 class  (darkMatterParticleClass), intent(inout)          :: self
 logical                         , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (darkMatterParticleClass), intent(inout)          :: self
 type   (inputParameters        ), intent(inout)          :: descriptor
 logical                         , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(darkMatterParticleClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (darkMatterParticleClass), intent(inout) :: self
 integer                         , intent(in   ) :: stateFile
 type   (fgsl_file               ), intent(in   ) :: fgslStateFile
 integer(c_size_t                ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (darkMatterParticleClass)                          , intent(inout) :: self
 type     (varying_string         ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                  )                          , intent(in   ) :: &
&  sourceName
end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(darkMatterParticleClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(darkMatterParticleClass), intent(inout) :: self
 class(darkMatterParticleClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (darkMatterParticleClass), intent(inout) :: self
 integer                         , intent(in   ) :: stateFile
 type   (fgsl_file               ), intent(in   ) :: fgslStateFile
 integer(c_size_t                ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`darkMatterParticleCDM` Provides a cold dark matter particle.

`darkMatterParticleWDMThermal` Provides a thermal warm dark matter particle.

Dark Matter Halo Profiles

Additional implementations for dark matter halo profiles are added using the `darkMatterProfile` class. The implementation should be placed in a file containing the directive:

```
!# <darkMatterProfile name="darkMatterProfileMyImplementation">
!# <description>A short description of the implementation.</description>
!# </darkMatterProfile>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `darkMatterProfileMyImplementation`. The file *must* define a type that extends the `darkMatterProfileClass` class (or extends another type which is itself an extension of the `darkMatterProfileClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`energy` Return the total energy for the given `node` in units of $M_\odot$ km^2 s^{-2}. Must have the following interface:

```
double precision function myImplementationEnergy(self,node)
 class(darkMatterProfileClass), intent(inout) :: self
 type (treeNode              ), intent(inout) :: node
end double precision function myImplementationEnergy
```

`circularVelocity` Returns the circular velocity (in km/s) in the dark matter profile of `node` at the given `radius` (given in units of Mpc). Must have the following interface:

```
double precision function myImplementationCircularVelocity(self,node,radius)
 class           (darkMatterProfileClass), intent(inout) :: self
 type            (treeNode              ), intent(inout) :: node
 double precision                        , intent(in   ) :: radius
end double precision function myImplementationCircularVelocity
```

`potential` Returns the gravitational potential (in (km/s)^2) in the dark matter profile of `node` at the given `radius` (given in units of Mpc). Must have the following interface:

```
double precision function myImplementationPotential(self,node,radius,status)
 class           (darkMatterProfileClass), intent(inout)           :: self
 type            (treeNode              ), intent(inout), pointer  :: node
 double precision                        , intent(in   )           :: radius
 integer                                 , intent(  out), optional :: status
end double precision function myImplementationPotential
```

`enclosedMass` Returns the enclosed mass (in $M_\odot$) in the dark matter profile of `node` at the given `radius` (given in units of Mpc). Must have the following interface:

```
double precision function myImplementationEnclosedMass(self,node,radius)
 class           (darkMatterProfileClass), intent(inout) :: self
 type            (treeNode              ), intent(inout) :: node
 double precision                        , intent(in   ) :: radius
end double precision function myImplementationEnclosedMass
```

`radiusEnclosingDensity` Returns the radius (in Mpc) enclosing a given density threshold (in $M_\odot$Mpc^{-3}) in the dark matter profile of `node`. Must have the following interface:

```
double precision function myImplementationRadiusEnclosingDensity(self,node,density)
 class           (darkMatterProfileClass), intent(inout)         :: self
 type            (treeNode              ), intent(inout), target :: node
 double precision                        , intent(in   )         :: density
end double precision function myImplementationRadiusEnclosingDensity
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(darkMatterProfileClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`radiusFromSpecificAngularMomentum` Returns the radius (in Mpc) in the dark matter profile of `node` at which the specific angular momentum of a circular orbit equals `specificAngularMomentum` (specified in units of km s^{-1} Mpc. Must have the following interface:

```
 double precision function myImplementationRadiusFromSpecificAngularMomentum(self,node,&
&  specificAngularMomentum)
  class           (darkMatterProfileClass), intent(inout)          :: self
  type            (treeNode              ), intent(inout), pointer :: node
  double precision                         , intent(in   )          :: &
&  specificAngularMomentum
 end double precision function myImplementationRadiusFromSpecificAngularMomentum
```

`freefallRadius` Returns the freefall radius (in Mpc) corresponding to the given `time` (in Gyr) in `node`. Must have the following interface:

```
double precision function myImplementationFreefallRadius(self,node,time)
 class           (darkMatterProfileClass), intent(inout) :: self
 type            (treeNode              ), intent(inout) :: node
 double precision                         , intent(in   ) :: time
end double precision function myImplementationFreefallRadius
```

`freeFallRadiusIncreaseRate` Returns the rate of increase of the freefall radius (in Mpc/Gyr) corresponding to the given `time` (in Gyr) in `node`. Must have the following interface:

```
double precision function myImplementationFreeFallRadiusIncreaseRate(self,node,time)
 class           (darkMatterProfileClass), intent(inout) :: self
 type            (treeNode              ), intent(inout) :: node
 double precision                         , intent(in   ) :: time
end double precision function myImplementationFreeFallRadiusIncreaseRate
```

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (darkMatterProfileClass), intent(inout)           :: self
  logical                        , intent(in   ), optional :: includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (darkMatterProfileClass), intent(inout)           :: self
 type   (inputParameters       ), intent(inout)           :: descriptor
 logical                        , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(darkMatterProfileClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (darkMatterProfileClass), intent(inout) :: self
 integer                        , intent(in   ) :: stateFile
 type   (fgsl_file             ), intent(in   ) :: fgslStateFile
 integer(c_size_t              ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (darkMatterProfileClass)                          , intent(inout) :: self
 type     (varying_string        ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                 )                          , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`energyGrowthRate` Return the rate of change of total energy for the given `node` in units of $M_\odot$ km^2 s^{-2} Gyr^{-1}. Must have the following interface:

```
double precision function myImplementationEnergyGrowthRate(self,node)
 class(darkMatterProfileClass), intent(inout) :: self
 type (treeNode              ), intent(inout) :: node
end double precision function myImplementationEnergyGrowthRate
```

`circularVelocityMaximum` Returns the maximum circular velocity (in km/s) in the dark matter profile of `node`. Must have the following interface:

```
double precision function myImplementationCircularVelocityMaximum(self,node)
 class(darkMatterProfileClass), intent(inout) :: self
 type (treeNode              ), intent(inout) :: node
end double precision function myImplementationCircularVelocityMaximum
```

`density` Returns the density (in $M_\odot$ Mpc^{-3}) in the dark matter profile of `node` at the given `radius` (given in units of Mpc). Must have the following interface:

```
double precision function myImplementationDensity(self,node,radius)
 class           (darkMatterProfileClass), intent(inout) :: self
 type            (treeNode              ), intent(inout) :: node
 double precision                        , intent(in   ) :: radius
end double precision function myImplementationDensity
```

`rotationNormalization` Returns the relation between specific angular momentum and rotation velocity (assuming a rotation velocity that is constant in radius) for the given `node`. Specifically, the normalization, A, returned is such that $V_{\rm rot} = AJ/M$ Must have the following interface:

```
double precision function myImplementationRotationNormalization(self,node)
 class(darkMatterProfileClass), intent(inout) :: self
 type (treeNode              ), intent(inout) :: node
end double precision function myImplementationRotationNormalization
```

`radiusEnclosingMass` Returns the radius (in Mpc) enclosing a given mass (in $M_\odot$) in the dark matter profile of `node`. A default implementation exists. If overridden the following interface must be used:

```
double precision function myImplementationRadiusEnclosingMass(self,node,mass)
 class           (darkMatterProfileClass), intent(inout)         :: self
 type            (treeNode              ), intent(inout), target :: node
 double precision                         , intent(in   )         :: mass
end double precision function myImplementationRadiusEnclosingMass
```

`radialVelocityDispersion` Returns the radial velocity dispersion (in km/s) in the dark matter profile of `node` at the given `radius` (given in units of Mpc). Must have the following interface:

```
double precision function myImplementationRadialVelocityDispersion(self,node,radius)
 class           (darkMatterProfileClass), intent(inout) :: self
 type            (treeNode              ), intent(inout) :: node
 double precision                         , intent(in   ) :: radius
end double precision function myImplementationRadialVelocityDispersion
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(darkMatterProfileClass), intent(inout) :: self
 class(darkMatterProfileClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (darkMatterProfileClass), intent(inout) :: self
 integer                        , intent(in   ) :: stateFile
 type   (fgsl_file             ), intent(in   ) :: fgslStateFile
 integer(c_size_t              ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

`densityLogSlope` Returns the logarithmic slope of the density profile in the dark matter profile of `node` at the given `radius` (given in units of Mpc). Must have the following interface:

```
double precision function myImplementationDensityLogSlope(self,node,radius)
 class           (darkMatterProfileClass), intent(inout) :: self
 type            (treeNode              ), intent(inout) :: node
 double precision                         , intent(in   ) :: radius
end double precision function myImplementationDensityLogSlope
```

`kSpace` Returns the normalized Fourier space density profile of the dark matter profile of `node` at the given `waveNumber` (given in units of Mpc^{-1}). Must have the following interface:

```
double precision function myImplementationKSpace(self,node,wavenumber)
 class           (darkMatterProfileClass), intent(inout)         :: self
 type            (treeNode              ), intent(inout), target :: node
 double precision                         , intent(in   )         :: wavenumber
end double precision function myImplementationKSpace
```

`radialMoment` Returns the $\mathtt{m}^{\mathrm{th}}$ radial moment of the dark matter profile of `node` optionally between the given `radiusMinimum` and `radiusMaximum` (given in units of Mpc). Must have the following interface:

```
 double precision function myImplementationRadialMoment(self,node,moment,radiusMinimum, &
&  radiusMaximum)
  class           (darkMatterProfileClass), intent(inout)          :: self
  type            (treeNode              ), intent(inout)          :: node
  double precision                         , intent(in   )          :: moment
  double precision                         , intent(in   ), optional :: radiusMinimum, &
&  radiusMaximum
 end double precision function myImplementationRadialMoment
```

Existing implementations are:

`darkMatterProfileDarkMatterOnly` An implementation of non-dark-matter-only dark matter halo profiles which are unchanged from their dark-matter-only counterpart.

`darkMatterProfileAdiabaticGnedin2004` AdiabaticGnedin2004 dark matter halo profiles.

Dark Matter Profile Concentrations

Additional implementations for dark matter profile concentrations are added using the `darkMatterProfileConcentration` class. The implementation should be placed in a file containing the directive:

```
!# <darkMatterProfileConcentration name="darkMatterProfileConcentrationMyImplementation">
!# <description>A short description of the implementation.</description>
!# </darkMatterProfileConcentration>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `darkMatterProfileConcentrationMyImplementation`. The file *must* define a type that extends the `darkMatterProfileConcentrationClass` class (or extends another type which is itself an extension of the `darkMatterProfileConcentrationClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`concentration` Returns the concentration parameter for the given `node`. Must have the following interface:

```
double precision function myImplementationConcentration(self,node)
 class(darkMatterProfileConcentrationClass), intent(inout)         :: self
 type (treeNode                           ), intent(inout), target :: node
end double precision function myImplementationConcentration
```

`concentrationMean` Returns the mean concentration parameter for a `node` of the given mass. A default implementation exists. If overridden the following interface must be used:

```
double precision function myImplementationConcentrationMean(self,node)
 class(darkMatterProfileConcentrationClass), intent(inout)         :: self
 type (treeNode                           ), intent(inout), target :: node
end double precision function myImplementationConcentrationMean
```

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (darkMatterProfileConcentrationClass), intent(inout)          :: self
  logical                                     , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (darkMatterProfileConcentrationClass), intent(inout)          :: self
 type   (inputParameters                    ), intent(inout)          :: descriptor
 logical                                     , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(darkMatterProfileConcentrationClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (darkMatterProfileConcentrationClass), intent(inout) :: self
 integer                                     , intent(in   ) :: stateFile
 type   (fgsl_file                           ), intent(in   ) :: fgslStateFile
 integer(c_size_t                            ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (darkMatterProfileConcentrationClass)                          , intent(&
&  inout) :: self
 type     (varying_string                     ), allocatable, dimension(:), intent(&
&  inout) :: allowedParameters
 character(len=*                              )                          , intent(in  &
&   ) :: sourceName
 end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(darkMatterProfileConcentrationClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(darkMatterProfileConcentrationClass), intent(inout) :: self
 class(darkMatterProfileConcentrationClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`densityContrastDefinition` Returns a `virialDensityContrast` object describing the virial density contrast used to define this concentration. Must have the following interface:

```
class(virialDensityContrastClass) function myImplementationDensityContrastDefinition(&
&  self)
 class(darkMatterProfileConcentrationClass), intent(inout) :: self
end class(virialDensityContrastClass) function &
&  myImplementationDensityContrastDefinition
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (darkMatterProfileConcentrationClass), intent(inout) :: self
 integer                                     , intent(in   ) :: stateFile
 type   (fgsl_file                           ), intent(in   ) :: fgslStateFile
 integer(c_size_t                            ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

`darkMatterProfileDMODefinition` Returns a `darkMatterProfileDMO` object describing the dark matter density profile used to define this concentration. Must have the following interface:

```
class(darkMatterProfileDMOClass) function &
&  myImplementationDarkMatterProfileDMODefinition(self)
 class(darkMatterProfileConcentrationClass), intent(inout) :: self
end class(darkMatterProfileDMOClass) function &
&  myImplementationDarkMatterProfileDMODefinition
```

Existing implementations are:

`darkMatterProfileConcentrationDuttonMaccio2014` Dark matter halo concentrations are computed using the algorithm of Dutton and Macciò [2014]. See §12.11.3.

`darkMatterProfileConcentrationWDM` Dark matter halo concentrations are computed using the modifier of Schneider et al. [2012].

`darkMatterProfileConcentrationCorrea2015` Dark matter halo concentrations are computed using the algorithm of Correa et al. [2015].

`darkMatterProfileConcentrationKlypin2015` Dark matter halo concentrations are computed using the algorithm of Klypin et al. [2014].

`darkMatterProfileConcentrationMunozCuartas2011` Dark matter halo concentrations are computed using the algorithm of Muñoz-Cuartas et al. [2011]. See §12.11.3.

`darkMatterProfileConcentrationDiemerKravtsov2014` Dark matter halo concentrations are computed using the algorithm of Diemer and Kravtsov [2014]. See §12.11.3.

`darkMatterProfileConcentrationSchneider2015` Dark matter halo concentrations are computed using the algorithm of Schneider [2015]. See §12.11.3.

`darkMatterProfileConcentrationPrada2011` Dark matter halo concentrations are computed using the algorithm of Prada et al. [2011]. See §12.11.3.

`darkMatterProfileConcentrationBullock2001` Dark matter halo concentrations are computed using the algorithm of Bullock et al. [2001].

`darkMatterProfileConcentrationNFW1996` Dark matter halo concentrations are computed using the algorithm of Navarro et al. [1996]. See §12.11.3.

`darkMatterProfileConcentrationZhao2009` Dark matter halo concentrations are computed using the algorithm of Zhao et al. [2009]. See §12.11.3.

`darkMatterProfileConcentrationGao2008` Dark matter halo concentrations are computed using the algorithm of Gao et al. [2008]. See §12.11.3.

`darkMatterProfileConcentrationLudlow2016Fit` Dark matter halo concentrations are computed using the fitting function of Ludlow et al. [2016].

Dark Matter Halo Profiles

Additional implementations for dark matter halo profiles are added using the `darkMatterProfileDMO` class. The implementation should be placed in a file containing the directive:

```
!# <darkMatterProfileDMO name="darkMatterProfileDMOMyImplementation">
!# <description>A short description of the implementation.</description>
!# </darkMatterProfileDMO>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `darkMatterProfileDMOMyImplementation`. The file *must* define a type that extends the `darkMatterProfileDMOClass` class (or extends another type which is itself an extension of the `darkMatterProfileDMOClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`energy` Return the total energy for the given `node` in units of $M_\odot$ km^2 s^{-2}. Must have the following interface:

```
double precision function myImplementationEnergy(self,node)
 class(darkMatterProfileDMOClass), intent(inout) :: self
 type (treeNode                 ), intent(inout) :: node
end double precision function myImplementationEnergy
```

`circularVelocity` Returns the circular velocity (in km/s) in the dark matter profile of `node` at the given `radius` (given in units of Mpc). Must have the following interface:

```
double precision function myImplementationCircularVelocity(self,node,radius)
 class           (darkMatterProfileDMOClass), intent(inout) :: self
 type            (treeNode                 ), intent(inout) :: node
 double precision                            , intent(in   ) :: radius
end double precision function myImplementationCircularVelocity
```

`potential` Returns the gravitational potential (in (km/s)^2) in the dark matter profile of `node` at the given `radius` (given in units of Mpc). Must have the following interface:

```
double precision function myImplementationPotential(self,node,radius,status)
 class           (darkMatterProfileDMOClass), intent(inout)           :: self
 type            (treeNode                 ), intent(inout), pointer  :: node
 double precision                           , intent(in   )           :: radius
 integer                                    , intent(  out), optional :: status
end double precision function myImplementationPotential
```

`enclosedMass` Returns the enclosed mass (in $M_\odot$) in the dark matter profile of `node` at the given `radius` (given in units of Mpc). Must have the following interface:

```
double precision function myImplementationEnclosedMass(self,node,radius)
 class           (darkMatterProfileDMOClass), intent(inout) :: self
 type            (treeNode                 ), intent(inout) :: node
 double precision                           , intent(in   ) :: radius
end double precision function myImplementationEnclosedMass
```

`radiusEnclosingDensity` Returns the radius (in Mpc) enclosing a given density threshold (in $M_\odot \mathrm{Mpc}^{-3}$) in the dark matter profile of `node`. Must have the following interface:

```
double precision function myImplementationRadiusEnclosingDensity(self,node,density)
 class           (darkMatterProfileDMOClass), intent(inout)         :: self
 type            (treeNode                 ), intent(inout), target :: node
 double precision                           , intent(in   )         :: density
end double precision function myImplementationRadiusEnclosingDensity
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(darkMatterProfileDMOClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`radiusFromSpecificAngularMomentum` Returns the radius (in Mpc) in the dark matter profile of `node` at which the specific angular momentum of a circular orbit equals `specificAngularMomentum` (specified in units of km s^{-1} Mpc. Must have the following interface:

```
double precision function myImplementationRadiusFromSpecificAngularMomentum(self,node,&
&  specificAngularMomentum)
 class           (darkMatterProfileDMOClass), intent(inout)          :: self
 type            (treeNode                 ), intent(inout), pointer :: node
 double precision                           , intent(in   )          :: &
&  specificAngularMomentum
 end double precision function myImplementationRadiusFromSpecificAngularMomentum
```

`freefallRadius` Returns the freefall radius (in Mpc) corresponding to the given `time` (in Gyr) in `node`. Must have the following interface:

```
double precision function myImplementationFreefallRadius(self,node,time)
 class           (darkMatterProfileDMOClass), intent(inout) :: self
 type            (treeNode                 ), intent(inout) :: node
 double precision                           , intent(in   ) :: time
end double precision function myImplementationFreefallRadius
```

`freeFallRadiusIncreaseRate` Returns the rate of increase of the freefall radius (in Mpc/Gyr) corresponding to the given `time` (in Gyr) in `node`. Must have the following interface:

```
double precision function myImplementationFreeFallRadiusIncreaseRate(self,node,time)
 class           (darkMatterProfileDMOClass), intent(inout) :: self
 type            (treeNode                 ), intent(inout) :: node
 double precision                           , intent(in   ) :: time
end double precision function myImplementationFreeFallRadiusIncreaseRate
```

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
 class  (darkMatterProfileDMOClass), intent(inout)           :: self
 logical                           , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (darkMatterProfileDMOClass), intent(inout)           :: self
 type   (inputParameters          ), intent(inout)           :: descriptor
 logical                           , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(darkMatterProfileDMOClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (darkMatterProfileDMOClass), intent(inout) :: self
 integer                           , intent(in   ) :: stateFile
 type   (fgsl_file                 ), intent(in   ) :: fgslStateFile
 integer(c_size_t                  ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (darkMatterProfileDMOClass)                          , intent(inout) :: self
 type     (varying_string           ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                    )                          , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`energyGrowthRate` Returns the rate of chance of the total energy of `node` in units of $M_\odot$ km^2 s^{-2} Gyr^{-1}. Must have the following interface:

```
double precision function myImplementationEnergyGrowthRate(self,node)
 class(darkMatterProfileDMOClass), intent(inout)         :: self
 type (treeNode                 ), intent(inout), target :: node
end double precision function myImplementationEnergyGrowthRate
```

`circularVelocityMaximum` Returns the maximum circular velocity (in km/s) in the dark matter profile of `node`. Must have the following interface:

```
double precision function myImplementationCircularVelocityMaximum(self,node)
 class(darkMatterProfileDMOClass), intent(inout) :: self
 type (treeNode                 ), intent(inout) :: node
end double precision function myImplementationCircularVelocityMaximum
```

`density` Returns the density (in $M_\odot$ Mpc^{-3}) in the dark matter profile of `node` at the given `radius` (given in units of Mpc). Must have the following interface:

```
double precision function myImplementationDensity(self,node,radius)
 class           (darkMatterProfileDMOClass), intent(inout) :: self
 type            (treeNode                 ), intent(inout) :: node
 double precision                           , intent(in   ) :: radius
end double precision function myImplementationDensity
```

`rotationNormalization` Returns the relation between specific angular momentum and rotation velocity (assuming a rotation velocity that is constant in radius) for the given `node`. Specifically, the normalization, A, returned is such that $V_{\rm rot} = AJ/M$ Must have the following interface:

```
double precision function myImplementationRotationNormalization(self,node)
 class(darkMatterProfileDMOClass), intent(inout) :: self
 type (treeNode                 ), intent(inout) :: node
end double precision function myImplementationRotationNormalization
```

`radiusEnclosingMass` Returns the radius (in Mpc) enclosing a given mass (in $M_\odot$) in the dark matter profile of `node`. A default implementation exists. If overridden the following interface must be used:

```
double precision function myImplementationRadiusEnclosingMass(self,node,mass)
 class           (darkMatterProfileDMOClass), intent(inout)         :: self
 type            (treeNode                 ), intent(inout), target :: node
 double precision                           , intent(in   )         :: mass
end double precision function myImplementationRadiusEnclosingMass
```

`radialVelocityDispersion` Returns the radial velocity dispersion (in km/s) in the dark matter profile of `node` at the given `radius` (given in units of Mpc). Must have the following interface:

```
double precision function myImplementationRadialVelocityDispersion(self,node,radius)
 class           (darkMatterProfileDMOClass), intent(inout) :: self
 type            (treeNode                 ), intent(inout) :: node
 double precision                           , intent(in   ) :: radius
end double precision function myImplementationRadialVelocityDispersion
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(darkMatterProfileDMOClass), intent(inout) :: self
 class(darkMatterProfileDMOClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (darkMatterProfileDMOClass), intent(inout) :: self
 integer                           , intent(in   ) :: stateFile
 type   (fgsl_file                ), intent(in   ) :: fgslStateFile
 integer(c_size_t                 ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

`densityLogSlope` Returns the logarithmic slope of the density profile in the dark matter profile of `node` at the given `radius` (given in units of Mpc). Must have the following interface:

```
double precision function myImplementationDensityLogSlope(self,node,radius)
 class           (darkMatterProfileDMOClass), intent(inout) :: self
 type            (treeNode                 ), intent(inout) :: node
 double precision                           , intent(in   ) :: radius
end double precision function myImplementationDensityLogSlope
```

`kSpace` Returns the normalized Fourier space density profile of the dark matter profile of `node` at the given `waveNumber` (given in units of Mpc^{-1}). Must have the following interface:

```
double precision function myImplementationKSpace(self,node,wavenumber)
 class           (darkMatterProfileDMOClass), intent(inout)         :: self
 type            (treeNode                 ), intent(inout), target :: node
 double precision                           , intent(in   )         :: wavenumber
end double precision function myImplementationKSpace
```

`radialMoment` Returns the $\mathtt{m}^{\mathrm{th}}$ radial moment of the dark matter profile of `node` optionally between the given `radiusMinimum` and `radiusMaximum` (given in units of Mpc). Must have the following interface:

```
double precision function myImplementationRadialMoment(self,node,moment,radiusMinimum, &
&  radiusMaximum)
 class           (darkMatterProfileDMOClass), intent(inout)           :: self
 type            (treeNode                 ), intent(inout)           :: node
 double precision                           , intent(in   )           :: moment
 double precision                           , intent(in   ), optional :: radiusMinimum,&
&   radiusMaximum
end double precision function myImplementationRadialMoment
```

Existing implementations are:

`darkMatterProfileDMONFW` Navarro et al. [1997] dark matter halo profiles

`darkMatterProfileDMOTruncated` truncated dark matter halo profiles.

`darkMatterProfileDMOBurkert` Burkert [1995] dark matter halo profiles

`darkMatterProfileDMOTruncatedExponential` exponentially truncated dark matter halo profiles Kazantzidis et al. [2006].

`darkMatterProfileDMOHeated` Heated dark matter halo profiles.

`darkMatterProfileDMOEinasto` "Einasto" dark matter halo profiles

`darkMatterProfileDMOIsothermal` Isothermal dark matter halo profiles

Dark Matter Profile Heating

Additional implementations for dark matter profile heating are added using the `darkMatterProfileHeating` class. The implementation should be placed in a file containing the directive:

```
!# <darkMatterProfileHeating name="darkMatterProfileHeatingMyImplementation">
!# <description>A short description of the implementation.</description>
!# </darkMatterProfileHeating>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `darkMatterProfileHeatingMyImplementation`. The file *must* define a type that extends the `darkMatterProfileHeatingClass` class (or extends another type which is itself an extension of the `darkMatterProfileHeatingClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(darkMatterProfileHeatingClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`specificEnergyGradient` The gradient of the specific energy of heating at the given `radius` in the given `node`. Must have the following interface:

```
double precision function myImplementationSpecificEnergyGradient(self,node,&
&  darkMatterProfileDMO_,radius)
  class           (darkMatterProfileHeatingClass), intent(inout) :: self
  type            (treeNode                     ), intent(inout) :: node
  class           (darkMatterProfileDMOClass    ), intent(inout) :: &
&  darkMatterProfileDMO_
  double precision                                 , intent(in   ) :: radius
end double precision function myImplementationSpecificEnergyGradient
```

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (darkMatterProfileHeatingClass), intent(inout)          :: self
  logical                               , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (darkMatterProfileHeatingClass), intent(inout)           :: self
 type   (inputParameters              ), intent(inout)           :: descriptor
 logical                               , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(darkMatterProfileHeatingClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (darkMatterProfileHeatingClass), intent(inout) :: self
 integer                               , intent(in   ) :: stateFile
 type   (fgsl_file                     ), intent(in   ) :: fgslStateFile
 integer(c_size_t                      ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`specificEnergy` The specific energy of heating at the given `radius` in the given `node`. Must have the following interface:

```
 double precision function myImplementationSpecificEnergy(self,node,&
&  darkMatterProfileDMO_,radius)
  class           (darkMatterProfileHeatingClass), intent(inout) :: self
  type            (treeNode                     ), intent(inout) :: node
  class           (darkMatterProfileDMOClass    ), intent(inout) :: &
&  darkMatterProfileDMO_
  double precision                                 , intent(in   ) :: radius
 end double precision function myImplementationSpecificEnergy
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
  class    (darkMatterProfileHeatingClass)                          , intent(inout) :: &
&  self
  type     (varying_string              ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
  character(len=*                       )                          , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(darkMatterProfileHeatingClass), intent(inout) :: self
 class(darkMatterProfileHeatingClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (darkMatterProfileHeatingClass), intent(inout) :: self
 integer                               , intent(in   ) :: stateFile
 type   (fgsl_file                     ), intent(in   ) :: fgslStateFile
 integer(c_size_t                      ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

`specificEnergyIsEverywhereZero` Returns true if the specific energy is zero everywhere in the given `node`. Must have the following interface:

```
logical function myImplementationSpecificEnergyIsEverywhereZero(self,node,&
&  darkMatterProfileDMO_)
 class(darkMatterProfileHeatingClass), intent(inout) :: self
 type (treeNode                     ), intent(inout) :: node
 class(darkMatterProfileDMOClass    ), intent(inout) :: darkMatterProfileDMO_
end logical function myImplementationSpecificEnergyIsEverywhereZero
```

Existing implementations are:

`darkMatterProfileHeatingSummation` A dark matter profile heating model which sums over other heat sources.

`darkMatterProfileHeatingTwoBodyRelaxation` A dark matter profile heating model which computes heating due to two-body relaxation.

`darkMatterProfileHeatingNull` A dark matter profile heating model in which the heating is always zero.

`darkMatterProfileHeatingTidal` A dark matter profile heating model which accounts for heating due to tidal shocking.

Dark Matter Profile Scale Radii

Additional implementations for dark matter profile scale radii are added using the `darkMatterProfileScaleRadius` class. The implementation should be placed in a file containing the directive:

```
!# <darkMatterProfileScaleRadius name="darkMatterProfileScaleRadiusMyImplementation">
!# <description>A short description of the implementation.</description>
!# </darkMatterProfileScaleRadius>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `darkMatterProfile-ScaleRadiusMyImplementation`. The file *must* define a type that extends the `darkMatterProfileScaleRadiusClass` class (or extends another type which is itself an extension of the `darkMatterProfileScaleRadiusClass`

class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (darkMatterProfileScaleRadiusClass), intent(inout)          :: self
  logical                                   , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (darkMatterProfileScaleRadiusClass), intent(inout)          :: self
 type   (inputParameters                  ), intent(inout)          :: descriptor
 logical                                   , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(darkMatterProfileScaleRadiusClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`radius` Returns the scale radius for the given `node`. Must have the following interface:

```
double precision function myImplementationRadius(self,node)
 class(darkMatterProfileScaleRadiusClass), intent(inout)         :: self
 type (treeNode                          ), intent(inout), target :: node
end double precision function myImplementationRadius
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (darkMatterProfileScaleRadiusClass), intent(inout) :: self
 integer                                   , intent(in   ) :: stateFile
 type   (fgsl_file                         ), intent(in   ) :: fgslStateFile
 integer(c_size_t                          ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (darkMatterProfileScaleRadiusClass)                          , intent(inout)&
&   :: self
```

```
  type     (varying_string                  ), allocatable, dimension(:), intent(inout)&
&   :: allowedParameters
  character(len=*                           )                           , intent(in   )&
&   :: sourceName
 end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(darkMatterProfileScaleRadiusClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(darkMatterProfileScaleRadiusClass), intent(inout) :: self
 class(darkMatterProfileScaleRadiusClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (darkMatterProfileScaleRadiusClass), intent(inout) :: self
 integer                                   , intent(in   ) :: stateFile
 type   (fgsl_file                         ), intent(in   ) :: fgslStateFile
 integer(c_size_t                          ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`darkMatterProfileScaleRadiusZero` Dark matter halo scale radii class in which are assumed to be zero.

`darkMatterProfileScaleRadiusBinary` A dark matter halo profile scale radii class which switches between two methods based on a filter.

`darkMatterProfileScaleRadiusLudlow2014` Dark matter halo scale radii are computed using the algorithm of Ludlow et al. [2014].

`darkMatterProfileScaleRadiusConcentration` Dark matter halo scale radii are computed from the concentration.

`darkMatterProfileScaleRadiusLudlow2016` Dark matter halo scale radii are computed using the algorithm of Ludlow et al. [2016]. While Ludlow et al. [2016] used $\Delta = 200\rho_{\rm crit}$ to define halos, their model actually predicts the scale radius, r_{-2}, rather than the concentration. Therefore, here we report that the Ludlow et al. [2016] concentrations are defined using the model's own virial density contrast definition — this ensures that the predicted scale radii are applied directly to model halos.

Dark Matter Profile Shapes

Additional implementations for dark matter profile shapes are added using the `darkMatterProfileShape` class. The implementation should be placed in a file containing the directive:

```
!# <darkMatterProfileShape name="darkMatterProfileShapeMyImplementation">
!# <description>A short description of the implementation.</description>
!# </darkMatterProfileShape>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `darkMatterProfileShapeMyImplementation`. The file *must* define a type that extends the `darkMatterProfileShapeClass` class (or extends another type which is itself an extension of the `darkMatterProfileShapeClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (darkMatterProfileShapeClass), intent(inout)           :: self
  logical                             , intent(in   ), optional :: includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationDescriptor(self,descriptor,includeMethod)
  class  (darkMatterProfileShapeClass), intent(inout)           :: self
  type   (inputParameters            ), intent(inout)           :: descriptor
  logical                             , intent(in   ), optional :: includeMethod
 end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAutoHook(self)
  class(darkMatterProfileShapeClass), intent(inout) :: self
 end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
  class  (darkMatterProfileShapeClass), intent(inout) :: self
  integer                             , intent(in   ) :: stateFile
  type   (fgsl_file                  ), intent(in   ) :: fgslStateFile
  integer(c_size_t                   ), intent(in   ) :: stateOperationID
 end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (darkMatterProfileShapeClass)                          , intent(inout) :: &
&  self
 type     (varying_string             ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                      )                          , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(darkMatterProfileShapeClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(darkMatterProfileShapeClass), intent(inout) :: self
 class(darkMatterProfileShapeClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`shape` Returns the shape parameter for the given `node`. Must have the following interface:

```
double precision function myImplementationShape(self,node)
 class(darkMatterProfileShapeClass), intent(inout)          :: self
 type (treeNode                   ), intent(inout), pointer :: node
end double precision function myImplementationShape
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (darkMatterProfileShapeClass), intent(inout) :: self
 integer                             , intent(in   ) :: stateFile
 type   (fgsl_file                   ), intent(in   ) :: fgslStateFile
 integer(c_size_t                    ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`darkMatterProfileShapeGao2008` Dark matter halo shape parameters are computed using the algorithm of Gao et al. [2008].

`darkMatterProfileShapeKlypin2015` Dark matter halo shape parameters are computed using the algorithm of Klypin et al. [2014].

One-dimensional Distribution Functions

Additional implementations for one-dimensional distribution functions are added using the `distributionFunction1D` class. The implementation should be placed in a file containing the directive:

```
!# <distributionFunction1D name="distributionFunction1DMyImplementation">
!# <description>A short description of the implementation.</description>
!# </distributionFunction1D>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `distributionFunction1DMyImplementation`. The file *must* define a type that extends the `distributionFunction1DClass` class (or extends another type which is itself an extension of the `distributionFunction1DClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`minimum` Returns the minimum possible value in the distribution. Must have the following interface:

```
double precision function myImplementationMinimum(self)
 class(distributionFunction1DClass), intent(inout) :: self
end double precision function myImplementationMinimum
```

`maximum` Returns the maximum possible value in the distribution. Must have the following interface:

```
double precision function myImplementationMaximum(self)
 class(distributionFunction1DClass), intent(inout) :: self
end double precision function myImplementationMaximum
```

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
 class  (distributionFunction1DClass), intent(inout)           :: self
 logical                             , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (distributionFunction1DClass), intent(inout)           :: self
 type   (inputParameters            ), intent(inout)           :: descriptor
 logical                             , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(distributionFunction1DClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (distributionFunction1DClass), intent(inout) :: self
 integer                             , intent(in   ) :: stateFile
 type   (fgsl_file                  ), intent(in   ) :: fgslStateFile
 integer(c_size_t                   ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (distributionFunction1DClass)                          , intent(inout) :: &
&  self
 type     (varying_string             ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                      )                          , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`samplerReset` Reset the sampler for the distribution. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationSamplerReset(self)
 class(distributionFunction1DClass), intent(inout) :: self
end subroutine myImplementationSamplerReset
```

`density` Return the probability density at `x`. Must have the following interface:

```
double precision function myImplementationDensity(self,x)
 class           (distributionFunction1DClass), intent(inout) :: self
 double precision                             , intent(in   ) :: x
end double precision function myImplementationDensity
```

`inverse` Return the value of the independent variable corresponding to cumulative probability `p`. Must have the following interface:

```
double precision function myImplementationInverse(self,p)
 class           (distributionFunction1DClass), intent(inout) :: self
 double precision                             , intent(in   ) :: p
end double precision function myImplementationInverse
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(distributionFunction1DClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(distributionFunction1DClass), intent(inout) :: self
 class(distributionFunction1DClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`sample` Return a random deviate from the distribution. A default implementation exists. If overridden the following interface must be used:

```
double precision function myImplementationSample(self,incrementSeed,ompThreadOffset, &
&  mpiRankOffset,randomNumberGenerator)
 class  (distributionFunction1DClass), intent(inout)           :: self
 integer                             , intent(in   ), optional :: incrementSeed
 logical                             , intent(in   ), optional :: ompThreadOffset, &
&  mpiRankOffset
 type   (pseudoRandom               ), intent(inout), optional :: randomNumberGenerator
end double precision function myImplementationSample
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (distributionFunction1DClass), intent(inout) :: self
 integer                             , intent(in   ) :: stateFile
 type   (fgsl_file                  ), intent(in   ) :: fgslStateFile
 integer(c_size_t                   ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

`cumulative` Return the cumulative probability at `x`. Must have the following interface:

```
double precision function myImplementationCumulative(self,x)
 class           (distributionFunction1DClass), intent(inout) :: self
 double precision                             , intent(in   ) :: x
end double precision function myImplementationCumulative
```

Existing implementations are:

`distributionFunction1DUniform` A uniform 1D distribution function class.

`distributionFunction1DBeta` A beta 1D distribution function class.

`distributionFunction1DVoight` A 1D distribution function class for Voight profiles.

`distributionFunction1DStudentT` A 1D Student-t distibution function.

`distributionFunction1DPeakBackground` A peakBackground 1D distribution function class.

`distributionFunction1DLogUniform` A 1D distibution function which is uniform in the logarithm of the variable.

`distributionFunction1DNormal` A normal 1D distribution function class.

`distributionFunction1DNegativeExponential` A negative exponential 1D distribution function class.

`distributionFunction1DCauchy` A 1D Cauchy distibution function.

`distributionFunction1DLogNormal` A normal 1D distribution function class.

`distributionFunction1DGamma` A 1D gamma distribution function class.

One-dimensional Discrete Distribution Functions

Additional implementations for one-dimensional discrete distribution functions are added using the `distributionFunctionDis` class. The implementation should be placed in a file containing the directive:

```
!# <distributionFunctionDiscrete1D name="distributionFunctionDiscrete1DMyImplementation">
!# <description>A short description of the implementation.</description>
!# </distributionFunctionDiscrete1D>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `distributionFunctionDiscrete1DMyImplementation`. The file *must* define a type that extends the `distributionFunctionDiscrete1DClass` class (or extends another type which is itself an extension of the `distributionFunctionDiscrete1DClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`minimum` Returns the minimum possible value in the distribution. Must have the following interface:

```
integer function myImplementationMinimum(self)
 class(distributionFunctionDiscrete1DClass), intent(inout) :: self
end integer function myImplementationMinimum
```

`maximum` Returns the maximum possible value in the distribution. Must have the following interface:

```
integer function myImplementationMaximum(self)
 class(distributionFunctionDiscrete1DClass), intent(inout) :: self
end integer function myImplementationMaximum
```

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (distributionFunctionDiscrete1DClass), intent(inout)           :: self
  logical                                     , intent(in   ), optional :: &
& includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (distributionFunctionDiscrete1DClass), intent(inout)           :: self
 type   (inputParameters                    ), intent(inout)           :: descriptor
 logical                                     , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`massLogarithmic` Return the logarithm of the probability mass at `x`. Must have the following interface:

```
double precision function myImplementationMassLogarithmic(self,x)
 class  (distributionFunctionDiscrete1DClass), intent(inout) :: self
 integer                                     , intent(in   ) :: x
end double precision function myImplementationMassLogarithmic
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(distributionFunctionDiscrete1DClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (distributionFunctionDiscrete1DClass), intent(inout) :: self
 integer                                     , intent(in   ) :: stateFile
 type   (fgsl_file                           ), intent(in   ) :: fgslStateFile
 integer(c_size_t                            ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (distributionFunctionDiscrete1DClass)                          , intent(&
&  inout) :: self
 type     (varying_string                     ), allocatable, dimension(:), intent(&
&  inout) :: allowedParameters
 character(len=*                              )                          , intent(in  &
&   ) :: sourceName
 end subroutine myImplementationAllowedParameters
```

`samplerReset` Reset the sampler for the distribution. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationSamplerReset(self)
 class(distributionFunctionDiscrete1DClass), intent(inout) :: self
end subroutine myImplementationSamplerReset
```

`inverse` Return the value of the independent variable corresponding to cumulative probability `p`. Must have the following interface:

```
integer function myImplementationInverse(self,p)
 class           (distributionFunctionDiscrete1DClass), intent(inout) :: self
 double precision                                     , intent(in   ) :: p
end integer function myImplementationInverse
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(distributionFunctionDiscrete1DClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`mass` Return the probability mass at `x`. Must have the following interface:

```
double precision function myImplementationMass(self,x)
 class  (distributionFunctionDiscrete1DClass), intent(inout) :: self
 integer                                     , intent(in   ) :: x
end double precision function myImplementationMass
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(distributionFunctionDiscrete1DClass), intent(inout) :: self
 class(distributionFunctionDiscrete1DClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`sample` Return a random deviate from the distribution. A default implementation exists. If overridden the following interface must be used:

```
integer function myImplementationSample(self,incrementSeed,ompThreadOffset, &
&  mpiRankOffset,randomNumberGenerator)
 class  (distributionFunctionDiscrete1DClass), intent(inout)           :: self
 integer                                     , intent(in   ), optional :: incrementSeed
 logical                                     , intent(in   ), optional :: &
&  ompThreadOffset, mpiRankOffset
 type   (pseudoRandom                       ), intent(inout), optional :: &
&  randomNumberGenerator
end integer function myImplementationSample
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (distributionFunctionDiscrete1DClass), intent(inout) :: self
 integer                                     , intent(in   ) :: stateFile
 type   (fgsl_file                          ), intent(in   ) :: fgslStateFile
 integer(c_size_t                           ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

`cumulative` Return the cumulative probability at `x`. Must have the following interface:

```
double precision function myImplementationCumulative(self,x)
 class  (distributionFunctionDiscrete1DClass), intent(inout) :: self
 integer                                     , intent(in   ) :: x
end double precision function myImplementationCumulative
```

Existing implementations are:

`distributionFunctionDiscrete1DNegativeBinomial` A negative binomial 1D discrete distribution function class.

`distributionFunctionDiscrete1DBinomial` A binomial 1D discrete distribution function class.

Evolve Forests Work Share

Additional implementations for evolve forests work share are added using the `evolveForestsWorkShare` class. The implementation should be placed in a file containing the directive:

```
!# <evolveForestsWorkShare name="evolveForestsWorkShareMyImplementation">
!# <description>A short description of the implementation.</description>
!# </evolveForestsWorkShare>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `evolveForestsWorkShareMyImplementation`. The file *must* define a type that extends the `evolveForestsWorkShareClass` class (or extends another type which is itself an extension of the `evolveForestsWorkShareClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`forestNumber` Return the number of the forest to process. Must have the following interface:

```
integer(c_size_t) function myImplementationForestNumber(self,utilizeOpenMPThreads)
 class  (evolveForestsWorkShareClass), intent(inout) :: self
 logical                             , intent(in   ) :: utilizeOpenMPThreads
end integer(c_size_t) function myImplementationForestNumber
```

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
 class  (evolveForestsWorkShareClass), intent(inout)           :: self
 logical                             , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (evolveForestsWorkShareClass), intent(inout)           :: self
 type   (inputParameters            ), intent(inout)           :: descriptor
 logical                             , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(evolveForestsWorkShareClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (evolveForestsWorkShareClass), intent(inout) :: self
 integer                             , intent(in   ) :: stateFile
 type   (fgsl_file                  ), intent(in   ) :: fgslStateFile
 integer(c_size_t                   ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (evolveForestsWorkShareClass)                          , intent(inout) :: &
&  self
 type     (varying_string             ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                      )                          , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`workerID` Return a unique worker ID. A default implementation exists. If overridden the following interface must be used:

```
integer function myImplementationWorkerID(self,utilizeOpenMPThreads)
 class  (evolveForestsWorkShareClass), intent(inout) :: self
 logical                             , intent(in   ) :: utilizeOpenMPThreads
end integer function myImplementationWorkerID
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(evolveForestsWorkShareClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(evolveForestsWorkShareClass), intent(inout) :: self
 class(evolveForestsWorkShareClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (evolveForestsWorkShareClass), intent(inout) :: self
 integer                             , intent(in   ) :: stateFile
 type   (fgsl_file                  ), intent(in   ) :: fgslStateFile
 integer(c_size_t                   ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

`workerCount` Return the count of workers. A default implementation exists. If overridden the following interface must be used:

```
integer function myImplementationWorkerCount(self,utilizeOpenMPThreads)
 class  (evolveForestsWorkShareClass), intent(inout) :: self
 logical                             , intent(in   ) :: utilizeOpenMPThreads
end integer function myImplementationWorkerCount
```

Existing implementations are:

`evolveForestsWorkShareCyclic` A forest evolution work sharing class in which forests are assigned by cycling through processes.

`evolveForestsWorkShareFCFS` A forest evolution work sharing class in which forests are assigned on a first-come-first-served basis.

`evolveForestsWorkShareStride` A forest evolution work sharing class in which forests are assigned by another work sharing class, but then strided over in steps of a specified size.

Excursion Set Barrier

Additional implementations for excursion set barrier are added using the `excursionSetBarrier` class. The implementation should be placed in a file containing the directive:

```
!# <excursionSetBarrier name="excursionSetBarrierMyImplementation">
!# <description>A short description of the implementation.</description>
!# </excursionSetBarrier>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `excursionSetBarrierMyImplementation`. The file *must* define a type that extends the `excursionSetBarrierClass` class (or extends another type which is itself an extension of the `excursionSetBarrierClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (excursionSetBarrierClass), intent(inout)           :: self
  logical                          , intent(in   ), optional :: includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (excursionSetBarrierClass), intent(inout)           :: self
 type   (inputParameters         ), intent(inout)           :: descriptor
 logical                          , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`barrier` Return the barrier height at the given variance and time. The `rateCompute` should be set to `true` if the barrier is being used in a calculation of barrier crossing rates, and to `false` otherwise. Must have the following interface:

```
double precision function myImplementationBarrier(self,variance   , time,node,&
&  rateCompute)
 class           (excursionSetBarrierClass), intent(inout) :: self
 double precision                          , intent(in   ) :: variance, time
 type            (treeNode                ), intent(inout) :: node
 logical                                   , intent(in   ) :: rateCompute
end double precision function myImplementationBarrier
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(excursionSetBarrierClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (excursionSetBarrierClass), intent(inout) :: self
 integer                          , intent(in   ) :: stateFile
 type   (fgsl_file               ), intent(in   ) :: fgslStateFile
 integer(c_size_t                ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (excursionSetBarrierClass)                          , intent(inout) :: self
 type     (varying_string          ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                   )                          , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(excursionSetBarrierClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(excursionSetBarrierClass), intent(inout) :: self
 class(excursionSetBarrierClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`barrierGradient` Return the gradient of the barrier with respect to variance at the given variance and time. The `rateCompute` should be set to `true` if the barrier is being used in a calculation of barrier crossing rates, and to `false` otherwise. Must have the following interface:

```
double precision function myImplementationBarrierGradient(self,variance   , time,node,&
&  rateCompute)
 class           (excursionSetBarrierClass), intent(inout) :: self
 double precision                          , intent(in   ) :: variance, time
 type            (treeNode                ), intent(inout) :: node
 logical                                   , intent(in   ) :: rateCompute
end double precision function myImplementationBarrierGradient
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (excursionSetBarrierClass), intent(inout) :: self
 integer                          , intent(in   ) :: stateFile
 type   (fgsl_file               ), intent(in   ) :: fgslStateFile
 integer(c_size_t                ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`excursionSetBarrierLinear` A linear excursion set barrier class.

`excursionSetBarrierCriticalOverdensity` A critical overdensity excursion set barrier class.

`excursionSetBarrierQuadratic` A quadratic excursion set barrier class.

`excursionSetBarrierRemapScale` An excursion set barrier class which remaps another class by multiplying by a constant.

`excursionSetBarrierRemapShethMoTormen` An excursion set barrier class which remaps another class using the Sheth et al. [2001] ellipsoidal collapse parameterization.

Excursion Set First Crossing Statistics

Additional implementations for excursion set first crossing statistics are added using the `excursionSetFirstCrossing` class. The implementation should be placed in a file containing the directive:

```
!# <excursionSetFirstCrossing name="excursionSetFirstCrossingMyImplementation">
!# <description>A short description of the implementation.</description>
!# </excursionSetFirstCrossing>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `excursionSetFirstCrossingMyImplementation`. The file *must* define a type that extends the `excursionSetFirstCrossingClass` class (or extends another type which is itself an extension of the `excursionSetFirstCrossingClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (excursionSetFirstCrossingClass), intent(inout)           :: self
  logical                                , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (excursionSetFirstCrossingClass), intent(inout)           :: self
 type   (inputParameters               ), intent(inout)           :: descriptor
 logical                                , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(excursionSetFirstCrossingClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (excursionSetFirstCrossingClass), intent(inout) :: self
 integer                                , intent(in   ) :: stateFile
 type   (fgsl_file                      ), intent(in   ) :: fgslStateFile
 integer(c_size_t                       ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`probability` Return the probability for a trajectory to make its first crossing of the barrier at the given `variance` and `time`. Must have the following interface:

```
double precision function myImplementationProbability(self,variance, time,node)
 class           (excursionSetFirstCrossingClass), intent(inout) :: self
 double precision                                , intent(in   ) :: variance, time
 type            (treeNode                      ), intent(inout) :: node
end double precision function myImplementationProbability
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (excursionSetFirstCrossingClass)                          , intent(inout) ::&
&   self
 type     (varying_string                ), allocatable, dimension(:), intent(inout) ::&
&   allowedParameters
 character(len=*                         )                          , intent(in   ) ::&
&   sourceName
 end subroutine myImplementationAllowedParameters
```

`coordinatedMPI` Sets the state of coordination under MPI. If set to true then the object can assume that any calculations it performs are being performed identically by all other MPI processes. This permits, for example, coordinated tabulation of results across MPI processes. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationCoordinatedMPI(self,state)
 class  (excursionSetFirstCrossingClass), intent(inout) :: self
 logical                                , intent(in   ) :: state
end subroutine myImplementationCoordinatedMPI
```

`rate` Return the rate of first crossing for excursion sets beginning at the given `variance` and `time` to transition to a first crossing at the given `varianceProgenitor`. Must have the following interface:

```
 double precision function myImplementationRate(self,variance, varianceProgenitor, time,&
&  node)
  class           (excursionSetFirstCrossingClass), intent(inout) :: self
  double precision                                , intent(in   ) :: variance, &
&  varianceProgenitor, time
  type            (treeNode                      ), intent(inout) :: node
 end double precision function myImplementationRate
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(excursionSetFirstCrossingClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(excursionSetFirstCrossingClass), intent(inout) :: self
 class(excursionSetFirstCrossingClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`rateNonCrossing` Return the rate of non-crossing for excursion sets beginning at the given `variance` and `time`. Must have the following interface:

```
double precision function myImplementationRateNonCrossing(self,variance, time,node)
 class           (excursionSetFirstCrossingClass), intent(inout) :: self
 double precision                                , intent(in   ) :: variance, time
 type            (treeNode                      ), intent(inout) :: node
end double precision function myImplementationRateNonCrossing
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (excursionSetFirstCrossingClass), intent(inout) :: self
 integer                                , intent(in   ) :: stateFile
 type   (fgsl_file                      ), intent(in   ) :: fgslStateFile
 integer(c_size_t                       ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`excursionSetFirstCrossingLinearBarrier` An excursion set first crossing statistics class for linear barriers.

`excursionSetFirstCrossingFarahiMidpoint` An excursion set first crossing statistics class using the algorithm of Benson et al. [2012], but using a midpoint method to perform the integrations [Du et al., 2017].

`excursionSetFirstCrossingZhangHui` An excursion set first crossing statistics class utilizing the algorithm of Zhang and Hui [2006].

`excursionSetFirstCrossingFarahi` An excursion set first crossing statistics class using the algorithm of Benson et al. [2012].

`excursionSetFirstCrossingZhangHuiHighOrder` An excursion set first crossing statistics class utilizing a higher order generalization of the algorithm of Zhang and Hui [2006].

Freefall radii.

Additional implementations for freefall radii. are added using the `freefallRadius` class. The implementation should be placed in a file containing the directive:

```
!# <freefallRadius name="freefallRadiusMyImplementation">
!# <description>A short description of the implementation.</description>
!# </freefallRadius>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `freefallRadiusMyImplementation`. The file *must* define a type that extends the `freefallRadiusClass` class (or extends another type which is itself an extension of the `freefallRadiusClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (freefallRadiusClass), intent(inout)           :: self
  logical                     , intent(in   ), optional :: includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (freefallRadiusClass), intent(inout)           :: self
 type   (inputParameters    ), intent(inout)           :: descriptor
 logical                     , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`radiusGrowthRate` Returns the rate of increase of the freefall radius for gas in the hot atmosphere surrounding the galaxy in `node` in units of Mpc/Gyr. Must have the following interface:

```
double precision function myImplementationRadiusGrowthRate(self,node)
 class(freefallRadiusClass), intent(inout) :: self
 type (treeNode           ), intent(inout) :: node
end double precision function myImplementationRadiusGrowthRate
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(freefallRadiusClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`radius` Returns the freefall radius for gas in the hot atmosphere surrounding the galaxy in `node` in units of Mpc. Must have the following interface:

```
double precision function myImplementationRadius(self,node)
 class(freefallRadiusClass), intent(inout) :: self
 type (treeNode           ), intent(inout) :: node
end double precision function myImplementationRadius
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (freefallRadiusClass), intent(inout) :: self
 integer                     , intent(in   ) :: stateFile
 type   (fgsl_file          ), intent(in   ) :: fgslStateFile
 integer(c_size_t           ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (freefallRadiusClass)                            , intent(inout) :: self
 type     (varying_string     ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*              )                            , intent(in   ) :: sourceName
end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(freefallRadiusClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(freefallRadiusClass), intent(inout) :: self
 class(freefallRadiusClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (freefallRadiusClass), intent(inout) :: self
 integer                     , intent(in   ) :: stateFile
 type   (fgsl_file          ), intent(in   ) :: fgslStateFile
 integer(c_size_t           ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`freefallRadiusDarkMatterHalo` A freefall radius class which computes the freefall radius based on the freefall time in the dark matter halo.

Freefall time available.

Additional implementations for freefall time available. are added using the `freefallTimeAvailable` class. The implementation should be placed in a file containing the directive:

```
!# <freefallTimeAvailable name="freefallTimeAvailableMyImplementation">
!# <description>A short description of the implementation.</description>
!# </freefallTimeAvailable>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `freefallTimeAvailableMyImplementation`. The file *must* define a type that extends the `freefallTimeAvailableClass` class (or extends another type which is itself an extension of the `freefallTimeAvailableClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
 class  (freefallTimeAvailableClass), intent(inout)          :: self
 logical                            , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (freefallTimeAvailableClass), intent(inout)          :: self
 type   (inputParameters           ), intent(inout)          :: descriptor
 logical                            , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(freefallTimeAvailableClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (freefallTimeAvailableClass), intent(inout) :: self
 integer                            , intent(in   ) :: stateFile
 type   (fgsl_file                 ), intent(in   ) :: fgslStateFile
 integer(c_size_t                  ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (freefallTimeAvailableClass)                          , intent(inout) :: &
&  self
 type     (varying_string           ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                    )                          , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`timeAvailable` Returns the time available for freefall in cooling calculations in `node`. Must have the following interface:

```
double precision function myImplementationTimeAvailable(self,node)
 class(freefallTimeAvailableClass), intent(inout) :: self
 type (treeNode                  ), intent(inout) :: node
end double precision function myImplementationTimeAvailable
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(freefallTimeAvailableClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(freefallTimeAvailableClass), intent(inout) :: self
 class(freefallTimeAvailableClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`timeAvailableIncreaseRate` Returns the rate at which the time available for freefall in cooling calculations increases in `node`. Must have the following interface:

```
double precision function myImplementationTimeAvailableIncreaseRate(self,node)
 class(freefallTimeAvailableClass), intent(inout) :: self
 type (treeNode                  ), intent(inout) :: node
end double precision function myImplementationTimeAvailableIncreaseRate
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (freefallTimeAvailableClass), intent(inout) :: self
 integer                            , intent(in   ) :: stateFile
 type   (fgsl_file                  ), intent(in   ) :: fgslStateFile
 integer(c_size_t                   ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`freefallTimeAvailableHaloFormation` A freefall time available class which computes the freefall time available based on the freefall time in the dark matter halo.

Bar instabilities in galactic disks

Additional implementations for bar instabilities in galactic disks are added using the `galacticDynamicsBarInstability` class. The implementation should be placed in a file containing the directive:

```
!# <galacticDynamicsBarInstability name="galacticDynamicsBarInstabilityMyImplementation">
!# <description>A short description of the implementation.</description>
!# </galacticDynamicsBarInstability>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `galacticDynamicsBarInstabilityMyImplementation`. The file *must* define a type that extends the `galacticDynamicsBarInstabilityClass` class (or extends another type which is itself an extension of the `galacticDynamicsBarInstabilityClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (galacticDynamicsBarInstabilityClass), intent(inout)           :: self
  logical                                     , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (galacticDynamicsBarInstabilityClass), intent(inout)           :: self
 type   (inputParameters                   ), intent(inout)           :: descriptor
 logical                                    , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(galacticDynamicsBarInstabilityClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (galacticDynamicsBarInstabilityClass), intent(inout) :: self
 integer                                     , intent(in   ) :: stateFile
 type   (fgsl_file                           ), intent(in   ) :: fgslStateFile
 integer(c_size_t                            ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (galacticDynamicsBarInstabilityClass)                          , intent(&
&  inout) :: self
 type     (varying_string                     ), allocatable, dimension(:), intent(&
&  inout) :: allowedParameters
 character(len=*                              )                          , intent(in  &
&   ) :: sourceName
 end subroutine myImplementationAllowedParameters
```

`timescale` Returns a timescale on which the bar instability depletes material from a disk into a pseudo-bulge. A negative value indicates no instability. Also returns the net torque due to any external force causing this instability. Must have the following interface:

```
subroutine myImplementationTimescale(self,node,timescale, externalDrivingSpecificTorque&
&  )
 class           (galacticDynamicsBarInstabilityClass), intent(inout) :: self
 type            (treeNode                           ), intent(inout) :: node
 double precision                                     , intent(  out) :: timescale, &
&  externalDrivingSpecificTorque
 end subroutine myImplementationTimescale
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(galacticDynamicsBarInstabilityClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(galacticDynamicsBarInstabilityClass), intent(inout) :: self
 class(galacticDynamicsBarInstabilityClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (galacticDynamicsBarInstabilityClass), intent(inout) :: self
 integer                                     , intent(in   ) :: stateFile
 type   (fgsl_file                           ), intent(in   ) :: fgslStateFile
 integer(c_size_t                            ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`galacticDynamicsBarInstabilityStable` A perfect stability model for galactic disk bar instability.

`galacticDynamicsBarInstabilityEfstathiou1982Tidal` The Efstathiou et al. [1982] model for galactic disk bar instability, but include the effects of tidal forces.

`galacticDynamicsBarInstabilityFixedTimescale` A simple model for galactic disk bar instability in which the timescale is fixed.

`galacticDynamicsBarInstabilityEfstathiou1982` The Efstathiou et al. [1982] model for galactic disk bar instability.

Galactic Filter

Additional implementations for galactic filter are added using the `galacticFilter` class. The implementation should be placed in a file containing the directive:

```
!# <galacticFilter name="galacticFilterMyImplementation">
!# <description>A short description of the implementation.</description>
!# </galacticFilter>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `galacticFilterMyImplementation`. The file *must* define a type that extends the `galacticFilterClass` class (or extends another type which is itself an extension of the `galacticFilterClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`passes` Return true if the given `node` passes the filter. Must have the following interface:

```
logical function myImplementationPasses(self,node)
 class(galacticFilterClass), intent(inout) :: self
 type (treeNode           ), intent(inout) :: node
end logical function myImplementationPasses
```

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (galacticFilterClass), intent(inout)           :: self
  logical                     , intent(in   ), optional :: includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationDescriptor(self,descriptor,includeMethod)
  class  (galacticFilterClass), intent(inout)           :: self
  type   (inputParameters    ), intent(inout)           :: descriptor
  logical                     , intent(in   ), optional :: includeMethod
 end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAutoHook(self)
  class(galacticFilterClass), intent(inout) :: self
 end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
  class  (galacticFilterClass), intent(inout) :: self
  integer                     , intent(in   ) :: stateFile
  type   (fgsl_file          ), intent(in   ) :: fgslStateFile
  integer(c_size_t           ), intent(in   ) :: stateOperationID
 end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
  class    (galacticFilterClass)                          , intent(inout) :: self
  type     (varying_string     ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
  character(len=*              )                          , intent(in   ) :: sourceName
 end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationObjectType(self)
  class(galacticFilterClass), intent(inout) :: self
 end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(galacticFilterClass), intent(inout) :: self
 class(galacticFilterClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (galacticFilterClass), intent(inout) :: self
 integer                     , intent(in   ) :: stateFile
 type   (fgsl_file          ), intent(in   ) :: fgslStateFile
 integer(c_size_t           ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`galacticFilterSpheroidStellarMass` A galactic high-pass filter for stellar mass. Galaxies with a spheroid stellar mass greater than or equal to a fixed threshold, $M_{\star,0} =$ `[massThreshold]`.

`galacticFilterBasicMass` A high-pass filter for basic mass. Halos with a basic mass mass greater than or equal to a fixed threshold, $M_0 =$ `[massThreshold]`.

`galacticFilterAlways` A galactic filter which always passes. (Used mostly for testing purposes.)

`galacticFilterStarFormationRate` A galactic high-pass filter for star formation rate. Galaxies with a combined disk plus spheroid star formation rate greater than or equal to a mass-dependent threshold. The threshold is given by

$$\log_{10}\left(\frac{\dot{\phi}_\mathrm{t}}{M_\odot\,\mathrm{Gyr}^{-1}}\right) = \alpha_0 + \alpha_1\left(\log_{10} M_\star - \log_{10} M_0\right), \tag{16.3}$$

where $M_0 =$ `[starFormationRateThresholdLogM0]`, $\alpha_0 =$ `[starFormationRateThresholdLogSFR0]`, and $\alpha_1 =$ `[starFormationRateThresholdLogSFR1]`.

`galacticFilterTreeHosted` A filter which passes only nodes that are hosted in a merger tree.

`galacticFilterStellarMassMorphology` A galactic high-pass filter for stellar mass-weighted morphology (i.e. spheroid-to-total ratio). Galaxies with a spheroid-to-total ratio (by stellar mass) greater than or equal to a fixed threshold, $R_{\star,0} =$ `[spheroidToTotalThreshold]`.

`galacticFilterNot` A filter which simply inverts the result of another filter.

`galacticFilterHaloIsolated` A filter which passes only isolated halos.

`galacticFilterAny` A galactic filter class which is the "any" combination of a set of other filters.

`galacticFilterLightcone` A galactic filter on lightcone geometry.

`galacticFilterStellarMass` A galactic high-pass filter for stellar mass. Galaxies with a combined disk plus spheroid stellar mass greater than or equal to a fixed threshold, $M_{\star,0} =$ `[massThreshold]`.

`galacticFilterRootNode` A filter which passes only nodes which are roots of their merger tree.

`galacticFilterNodeMajorMergerRecent` A low-pass filter for time since the last major node merger. Halos with a time of the last major node merger greater than or equal to the current time minus $\Delta t =$ `[timeRecent]` are passed.

`galacticFilterAll` A galactic filter class which is the "all" combination of a set of other filters.

`galacticFilterMainBranch` A filter which passes only main branch halos.

`galacticFilterHaloMass` A high-pass filter for basic mass. Halos with a halo mass greater than or equal to a fixed threshold, $M_0 =$ `[massThreshold]`.

`galacticFilterISMMass` A galactic high-pass filter for ISM mass. Galaxies with a combined disk plus spheroid ISM mass greater than or equal to a fixed threshold, $M_{\rm ISM,0} =$ `[massThreshold]`.

`galacticFilterSurveyGeometry` A filter which passes only nodes that lie within a survey geometry.

`galacticFilterStellarApparentMagnitudes` A galactic low-pass (i.e. bright-pass) filter for stellar apparent magnitudes. Galaxies with apparent magnitude in each band, i, less than or equal to a fixed threshold, $m_{0,i} =$ `[apparentMagnitudeThreshold]`.

`galacticFilterFormationTime` A filter which removes recently-formed halos. Halos with a formation time greater than the current time minus $\Delta t =$ `[timeRecent]` are removed.

`galacticFilterHostMassRange` Passes nodes with host halo basic mass, $M_{\rm host}$, in the range `[massMinimum]` $\leq M_{\rm host} <$ `[massMaximum]`.

`galacticFilterStellarAbsoluteMagnitudes` A galactic low-pass (i.e. bright-pass) filter for stellar absolute magnitudes. Galaxies with absolute magnitude in each band, i, less than or equal to a fixed threshold, $M_{0,i} =$ `[absoluteMagnitudeThreshold]`.

`galacticFilterHaloNotIsolated` A filter which passes only non-isolated halos.

Solvers for galactic structure

Additional implementations for solvers for galactic structure are added using the `galacticStructureSolver` class. The implementation should be placed in a file containing the directive:

```
!# <galacticStructureSolver name="galacticStructureSolverMyImplementation">
!# <description>A short description of the implementation.</description>
!# </galacticStructureSolver>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `galacticStructureSolverMyImplementation`. The file *must* define a type that extends the `galacticStructureSolverClass` class (or extends another type which is itself an extension of the `galacticStructureSolverClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (galacticStructureSolverClass), intent(inout)           :: self
  logical                              , intent(in   ), optional :: includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (galacticStructureSolverClass), intent(inout)           :: self
 type   (inputParameters             ), intent(inout)           :: descriptor
 logical                              , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`solve` Solves for the structure of components in the given `node`. Must have the following interface:

```
subroutine myImplementationSolve(self,node)
 class(galacticStructureSolverClass), intent(inout)         :: self
 type (treeNode                    ), intent(inout), target :: node
end subroutine myImplementationSolve
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(galacticStructureSolverClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (galacticStructureSolverClass), intent(inout) :: self
 integer                              , intent(in   ) :: stateFile
 type   (fgsl_file                   ), intent(in   ) :: fgslStateFile
 integer(c_size_t                    ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (galacticStructureSolverClass)                          , intent(inout) :: &
&  self
 type     (varying_string              ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                       )                          , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(galacticStructureSolverClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(galacticStructureSolverClass), intent(inout) :: self
 class(galacticStructureSolverClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`revert` Revert the structure of components in the given `node` (if necessary to ensure that the structure solver will give the same result when called consecutively). Must have the following interface:

```
subroutine myImplementationRevert(self,node)
 class(galacticStructureSolverClass), intent(inout) :: self
 type (treeNode                    ), intent(inout) :: node
end subroutine myImplementationRevert
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (galacticStructureSolverClass), intent(inout) :: self
 integer                              , intent(in   ) :: stateFile
 type   (fgsl_file                    ), intent(in   ) :: fgslStateFile
 integer(c_size_t                     ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`galacticStructureSolverLinear` A "linear" solver for galactic structure (no self-gravity of baryons, and size simply scales in proportion to specific angular momentum).

`galacticStructureSolverEquilibrium` An "equilibrium" solver for galactic structure.

`galacticStructureSolverFixed` A "fixed" solver for galactic structure (no self-gravity of baryons, and size simply scales in proportion to specific angular momentum).

`galacticStructureSolverSimple` A simple solver for galactic structure (self-gravity of baryons is ignored).

Gaunt Factors

Additional implementations for gaunt factors are added using the `gauntFactor` class. The implementation should be placed in a file containing the directive:

```
!# <gauntFactor name="gauntFactorMyImplementation">
!# <description>A short description of the implementation.</description>
!# </gauntFactor>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `gauntFactorMyImplementation`. The file *must* define a type that extends the `gauntFactorClass` class (or extends another type which is itself an extension of the `gauntFactorClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
 class  (gauntFactorClass), intent(inout)          :: self
 logical                  , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (gauntFactorClass), intent(inout)          :: self
 type   (inputParameters ), intent(inout)          :: descriptor
 logical                  , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(gauntFactorClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (gauntFactorClass), intent(inout) :: self
 integer                  , intent(in   ) :: stateFile
 type   (fgsl_file       ), intent(in   ) :: fgslStateFile
 integer(c_size_t        ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (gauntFactorClass)                          , intent(inout) :: self
 type     (varying_string  ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*           )                          , intent(in   ) :: sourceName
end subroutine myImplementationAllowedParameters
```

`total` Returns the thermally averaged, total Gaunt factor. Must have the following interface:

```
 double precision function myImplementationTotal(self,atomicNumber, electronNumber,&
&  temperature)
  class           (gauntFactorClass), intent(inout) :: self
  integer                           , intent(in   ) :: atomicNumber, electronNumber
  double precision                  , intent(in   ) :: temperature
 end double precision function myImplementationTotal
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(gauntFactorClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(gauntFactorClass), intent(inout) :: self
 class(gauntFactorClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (gauntFactorClass), intent(inout) :: self
 integer                  , intent(in   ) :: stateFile
 type   (fgsl_file       ), intent(in   ) :: fgslStateFile
 integer(c_size_t        ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`gauntFactorSutherland1998` Gaunt factors are computed using the fitting function of Sutherland [1998].

`gauntFactorVanHoof2014` Gaunt factors are computed using the fitting function of van Hoof et al. [2014].

Lightcone Geometries

Additional implementations for lightcone geometries are added using the `geometryLightcone` class. The implementation should be placed in a file containing the directive:

```
!# <geometryLightcone name="geometryLightconeMyImplementation">
!# <description>A short description of the implementation.</description>
!# </geometryLightcone>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `geometryLightcone-MyImplementation`. The file *must* define a type that extends the `geometryLightconeClass` class (or extends another type which is itself an extension of the `geometryLightconeClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
 class  (geometryLightconeClass), intent(inout)           :: self
 logical                        , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (geometryLightconeClass), intent(inout)           :: self
 type   (inputParameters       ), intent(inout)           :: descriptor
 logical                        , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(geometryLightconeClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (geometryLightconeClass), intent(inout) :: self
 integer                        , intent(in   ) :: stateFile
 type   (fgsl_file              ), intent(in   ) :: fgslStateFile
 integer(c_size_t               ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`isInLightcone` Returns true if the provided node lies within the lightcone. Must have the following interface:

```
logical function myImplementationIsInLightcone(self,node,atPresentEpoch,radiusBuffer)
 class           (geometryLightconeClass), intent(inout)           :: self
 type            (treeNode              ), intent(inout)           :: node
 logical                                 , intent(in   ), optional :: atPresentEpoch
 double precision                        , intent(in   ), optional :: radiusBuffer
end logical function myImplementationIsInLightcone
```

`position` Returns the position vector of a `node` (in units of Mpc) in the lightcone coordinate system. Must have the following interface:

```
double precision, dimension(3) function myImplementationPosition(self,node,instance)
 class  (geometryLightconeClass), intent(inout) :: self
 type   (treeNode               ), intent(inout) :: node
 integer(c_size_t               ), intent(in   ) :: instance
end double precision, dimension(3) function myImplementationPosition
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (geometryLightconeClass)                          , intent(inout) :: self
 type     (varying_string        ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                 )                          , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`solidAngle` Returns the solid angle subtended by the lightcone (in units of steradians). Must have the following interface:

```
double precision function myImplementationSolidAngle(self)
 class(geometryLightconeClass), intent(inout) :: self
end double precision function myImplementationSolidAngle
```

`velocity` Returns the velocity vector of a `node` (in units of km/s) in the lightcone coordinate system. Must have the following interface:

```
double precision, dimension(3) function myImplementationVelocity(self,node,instance)
 class  (geometryLightconeClass), intent(inout) :: self
 type   (treeNode              ), intent(inout) :: node
 integer(c_size_t              ), intent(in   ) :: instance
end double precision, dimension(3) function myImplementationVelocity
```

`replicationCount` Returns the number of times the given nodes appears in the lightone . Must have the following interface:

```
integer(c_size_t) function myImplementationReplicationCount(self,node)
 class(geometryLightconeClass), intent(inout) :: self
 type (treeNode              ), intent(inout) :: node
end integer(c_size_t) function myImplementationReplicationCount
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(geometryLightconeClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(geometryLightconeClass), intent(inout) :: self
 class(geometryLightconeClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (geometryLightconeClass), intent(inout) :: self
 integer                        , intent(in   ) :: stateFile
 type   (fgsl_file             ), intent(in   ) :: fgslStateFile
 integer(c_size_t              ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`geometryLightconeSquare` A lightcone geometry class which assumes a square field of view., i.e. defined such that a point (x, y, z) is in the survey angular mask if $|\mathrm{atan2}(y, x)| < \psi/2$ and $|\mathrm{atan2}(z, x)| < \psi/2$ where atan2() is the quadrant-aware inverse tangent function, and ψ is the angular size of the field, we compute the solid angle of the lightcone as follows. Define a spherical coodinate system (θ, ϕ) with the pole ($\theta = 0$) aligned with the x-axis. The solid angle of the field is then

$$\Omega = 2\pi \int_0^{\psi/2} \sin\theta \mathrm{d}\theta + 8 \int_{\psi/2}^{\tan^{-1}(\sqrt{2}\tan(\psi/2))} \mathrm{d}\theta \sin\theta \int_{\cos^{-1}(\tan(\psi/2)/\tan\theta)}^{\pi/4} \mathrm{d}\phi, \tag{16.4}$$

which is

$$\Omega = 2\pi[1 - \cos(\psi/2)] + 8 \int_{\psi/2}^{\tan^{-1}(\sqrt{2}\tan(\psi/2))} \mathrm{d}\theta \sin\theta \left[\frac{\pi}{4} - \cos^{-1}\left(\frac{\tan(\psi/2)}{\tan\theta}\right)\right], \tag{16.5}$$

or

$$\Omega = 2\pi[1 - \cos(\tan^{-1}(\sqrt{2}\tan(\psi/2)))] - 8 \int_{\psi/2}^{\tan^{-1}(\sqrt{2}\tan(\psi/2))} \mathrm{d}\theta \sin\theta \cos^{-1}\left(\frac{\tan(\psi/2)}{\tan\theta}\right), \tag{16.6}$$

The final integral can be evaluated (using Mathematica for example) to give

$$\begin{aligned}\Omega \;=\; & 2\pi[3 - \cos(\tan^{-1}(\sqrt{2}\tan(\psi/2)))] - 8\sin(x)\Big(\sqrt{(a^2+1)\cos(2x)+a^2-1}(\log(a(\sqrt{2}\sqrt{2a^2\cos^2(x)+\cos(2x)-1} \\ & +2a)) - \log(\sqrt{\cos(2x)-1}))\sqrt{\csc^2(x)(-((a^2+1)\cos(2x)+a^2-1))} - \cot(x)((a^2+1)\cos(2x)+a^2-1) \\ & \cos^{-1}(a\cot(x))\Big)/[(a^2+1)\cos(2x)+a^2-1],\end{aligned} \tag{16.7}$$

where $a = \tan(\psi/2)$ and $x = \tan^{-1}[\sqrt{2}\tan(\psi/2)]$. Various sub-parameters specify the details of the light geometry. The `lengthReplication` parameter should give the length of the simulation box (the box will be replicated to span the volume covered by the lightcone), with the `lengthUnitsInSI` parameter giving the length unit in SI units and `lengthHubbleExponent` giving the exponent of h that appears in the length unit. The `angularSize` parameter of `fieldOfView` should gives the length of the side of the square field of view in degrees. The `origin` element must contain the x, y, z coordinates of the origin of the lightcone within the simulation box, while the `unitVectorX` parameters must give unit vectors which point along the lightcone (for `X`= 1), and in the two directions perpendicular to the lightcone (for `X`= 2 and 3). The `redshift` parameters must list the redshifts of available outputs.

Gravitational Lensing

Additional implementations for gravitational lensing are added using the `gravitationalLensing` class. The implementation should be placed in a file containing the directive:

```
!# <gravitationalLensing name="gravitationalLensingMyImplementation">
!# <description>A short description of the implementation.</description>
!# </gravitationalLensing>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `gravitationalLensingMyImplementation`. The file *must* define a type that extends the `gravitationalLensingClass` class (or extends another type which is itself an extension of the `gravitationalLensingClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
 class  (gravitationalLensingClass), intent(inout)          :: self
 logical                           , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (gravitationalLensingClass), intent(inout)          :: self
 type   (inputParameters          ), intent(inout)          :: descriptor
 logical                           , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(gravitationalLensingClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (gravitationalLensingClass), intent(inout) :: self
 integer                           , intent(in   ) :: stateFile
 type   (fgsl_file                 ), intent(in   ) :: fgslStateFile
 integer(c_size_t                  ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`magnificationCDF` Returns the cumulative probability function for magnification. Must have the following interface:

```
double precision function myImplementationMagnificationCDF(self,magnification, redshift&
&  , scaleSource)
 class           (gravitationalLensingClass), intent(inout) :: self
 double precision                           , intent(in   ) :: magnification, redshift,&
&   scaleSource
end double precision function myImplementationMagnificationCDF
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (gravitationalLensingClass)                          , intent(inout) :: self
 type     (varying_string          ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                   )                          , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`magnificationPDF` Returns the differential probability function for magnification. Must have the following interface:

```
 double precision function myImplementationMagnificationPDF(self,magnification, redshift&
&  , scaleSource)
  class           (gravitationalLensingClass), intent(inout) :: self
  double precision                           , intent(in   ) :: magnification, redshift,&
&   scaleSource
 end double precision function myImplementationMagnificationPDF
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(gravitationalLensingClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(gravitationalLensingClass), intent(inout) :: self
 class(gravitationalLensingClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (gravitationalLensingClass), intent(inout) :: self
 integer                           , intent(in   ) :: stateFile
 type   (fgsl_file                 ), intent(in   ) :: fgslStateFile
 integer(c_size_t                  ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`gravitationalLensingBaryonicModifier` Implements the gravitational lensing distribution by modifying another distribution for the effects of baryons.

`gravitationalLensingTakahashi2011` Implements the gravitational lensing distributions of Takahashi et al. [2011]. See §12.19.1.

Halo Environment

Additional implementations for halo environment are added using the `haloEnvironment` class. The implementation should be placed in a file containing the directive:

```
!# <haloEnvironment name="haloEnvironmentMyImplementation">
!# <description>A short description of the implementation.</description>
!# </haloEnvironment>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `haloEnvironmentMyImplementation`. The file *must* define a type that extends the `haloEnvironmentClass` class (or extends another type which is itself an extension of the `haloEnvironmentClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`overdensityLinear` Return the environmental linear overdensity for the given `node`. Must have the following interface:

```
double precision function myImplementationOverdensityLinear(self,node,presentDay)
 class  (haloEnvironmentClass), intent(inout)           :: self
 type   (treeNode            ), intent(inout)           :: node
 logical                      , intent(in   ), optional :: presentDay
end double precision function myImplementationOverdensityLinear
```

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
 class  (haloEnvironmentClass), intent(inout)           :: self
 logical                      , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (haloEnvironmentClass), intent(inout)           :: self
 type   (inputParameters     ), intent(inout)           :: descriptor
 logical                      , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`overdensityNonLinear` Return the environmental non-linear overdensity for the given `node`. Must have the following interface:

```
double precision function myImplementationOverdensityNonLinear(self,node)
 class(haloEnvironmentClass), intent(inout) :: self
 type (treeNode            ), intent(inout) :: node
end double precision function myImplementationOverdensityNonLinear
```

`overdensityLinearMinimum` Return the minimum linear overdensity for which the environmental overdensity PDF is non-zero. A default implementation exists. If overridden the following interface must be used:

```
double precision function myImplementationOverdensityLinearMinimum(self)
 class(haloEnvironmentClass), intent(inout) :: self
end double precision function myImplementationOverdensityLinearMinimum
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(haloEnvironmentClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`overdensityLinearMaximum` Return the maximum linear overdensity for which the environmental overdensity PDF is non-zero. A default implementation exists. If overridden the following interface must be used:

```
double precision function myImplementationOverdensityLinearMaximum(self)
 class(haloEnvironmentClass), intent(inout) :: self
end double precision function myImplementationOverdensityLinearMaximum
```

`pdf` Return the PDF of the environmental overdensity for the given overdensity. Must have the following interface:

```
double precision function myImplementationPdf(self,overdensity)
 class           (haloEnvironmentClass), intent(inout) :: self
 double precision                      , intent(in   ) :: overdensity
end double precision function myImplementationPdf
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (haloEnvironmentClass), intent(inout) :: self
 integer                      , intent(in   ) :: stateFile
 type   (fgsl_file           ), intent(in   ) :: fgslStateFile
 integer(c_size_t            ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`environmentRadius` Return the radius of the region used to defined the environmental. Must have the following interface:

```
double precision function myImplementationEnvironmentRadius(self)
 class(haloEnvironmentClass), intent(inout) :: self
end double precision function myImplementationEnvironmentRadius
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (haloEnvironmentClass)                          , intent(inout) :: self
 type     (varying_string      ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
```

```
 character(len=*               )                              , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`environmentMass` Return the mean mass contained in the region used to defined the environmental. Must have the following interface:

```
double precision function myImplementationEnvironmentMass(self)
 class(haloEnvironmentClass), intent(inout) :: self
end double precision function myImplementationEnvironmentMass
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(haloEnvironmentClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(haloEnvironmentClass), intent(inout) :: self
 class(haloEnvironmentClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (haloEnvironmentClass), intent(inout) :: self
 integer                      , intent(in   ) :: stateFile
 type   (fgsl_file           ), intent(in   ) :: fgslStateFile
 integer(c_size_t            ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

`overdensityLinearSet` Set the environmental overdensity for the give node. Must have the following interface:

```
subroutine myImplementationOverdensityLinearSet(self,node,overdensity)
 class           (haloEnvironmentClass), intent(inout) :: self
 type            (treeNode            ), intent(inout) :: node
 double precision                      , intent(in   ) :: overdensity
end subroutine myImplementationOverdensityLinearSet
```

`cdf` Return the CDF of the environmental overdensity for the given overdensity. Must have the following interface:

```
double precision function myImplementationCdf(self,overdensity)
 class           (haloEnvironmentClass), intent(inout) :: self
 double precision                      , intent(in   ) :: overdensity
end double precision function myImplementationCdf
```

Existing implementations are:

`haloEnvironmentUniform` Implements a uniform halo environment.

`haloEnvironmentNormal` Implements a normally-distributed halo environment.

`haloEnvironmentLogNormal` Implements a log-normal halo environment.

Halo Mass Function

Additional implementations for halo mass function are added using the `haloMassFunction` class. The implementation should be placed in a file containing the directive:

```
!# <haloMassFunction name="haloMassFunctionMyImplementation">
!# <description>A short description of the implementation.</description>
!# </haloMassFunction>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `haloMassFunctionMyImplementation`. The file *must* define a type that extends the `haloMassFunctionClass` class (or extends another type which is itself an extension of the `haloMassFunctionClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`massFraction` Return the halo mass fraction at `time` [Gyr] integrated between `massLow` and `massHigh` [$M_\odot$]. A default implementation exists. If overridden the following interface must be used:

```
 double precision function myImplementationMassFraction(self,time, massLow, massHigh,&
&  node)
  class           (haloMassFunctionClass), intent(inout)                  :: self
  double precision                       , intent(in   )                  :: time, &
&  massLow, massHigh
  type            (treeNode             ), intent(inout), optional, target :: node
 end double precision function myImplementationMassFraction
```

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (haloMassFunctionClass), intent(inout)           :: self
  logical                       , intent(in   ), optional :: includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationDescriptor(self,descriptor,includeMethod)
  class  (haloMassFunctionClass), intent(inout)           :: self
  type   (inputParameters      ), intent(inout)           :: descriptor
  logical                       , intent(in   ), optional :: includeMethod
 end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(haloMassFunctionClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (haloMassFunctionClass), intent(inout) :: self
 integer                       , intent(in   ) :: stateFile
 type   (fgsl_file            ), intent(in   ) :: fgslStateFile
 integer(c_size_t             ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`integrated` Return the halo mass function at `time` [Gyr] integrated between `massLow` and `massHigh` [$M_\odot$]. A default implementation exists. If overridden the following interface must be used:

```
double precision function myImplementationIntegrated(self,time, massLow, massHigh,node)
 class           (haloMassFunctionClass), intent(inout)                  :: self
 double precision                       , intent(in   )                  :: time, &
&  massLow, massHigh
 type            (treeNode             ), intent(inout), optional, target :: node
end double precision function myImplementationIntegrated
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (haloMassFunctionClass)                          , intent(inout) :: self
 type     (varying_string       ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                )                          , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(haloMassFunctionClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(haloMassFunctionClass), intent(inout) :: self
 class(haloMassFunctionClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (haloMassFunctionClass), intent(inout) :: self
 integer                       , intent(in   ) :: stateFile
 type   (fgsl_file            ), intent(in   ) :: fgslStateFile
 integer(c_size_t             ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

`differential` Return the differential halo mass function for `mass` [$M_\odot$] at `time` [Gyr]. Must have the following interface:

```
double precision function myImplementationDifferential(self,time, mass,node)
 class           (haloMassFunctionClass), intent(inout)           :: self
 double precision                       , intent(in   )           :: time, mass
 type            (treeNode             ), intent(inout), optional :: node
end double precision function myImplementationDifferential
```

Existing implementations are:

`haloMassFunctionRodriguezPuebla2016` The halo mass function is computed from the function given by Tinker et al. [2008], and using the fits of Rodríguez-Puebla et al. [2016] for the parameter values.

`haloMassFunctionEnvironmental` The halo mass function is computed by handling the transition though the environment mass scale.

`haloMassFunctionErrorConvolved` The halo mass function is computed by convolving another halo mass function with a mass dependent error. Specifically, the mass function is convolved with a Gaussian random error distribution with width computed using the given `nbodyHaloMassError` object.

`haloMassFunctionFofBias` The halo mass function is computed by modifying another halo mass function to mimic systematic errors arising in the friends-of-friends halo finding algorithm. Specifically, a systematic shift in mass motivated by the results of percolation theory [More et al., 2011, their eqn. B11] is applied. In particular, $M_{\rm particle} =$`[massParticle]` is the mass of the particle in the simulation to which the friends-of-friends algorithm was applied.

`haloMassFunctionBhattacharya2011` The halo mass function is computed from the function given by Bhattacharya et al. [2011].

`haloMassFunctionTinker2008` The halo mass function is computed from the function given by Tinker et al. [2008], and using their fits for the parameter values. See §12.6.11.

`haloMassFunctionDespali2015` The halo mass function is computed from the function given by Despali et al. [2015]. See §12.6.11.

`haloMassFunctionPressSchechter` The halo mass function is computed from the function given by Press and Schechter [1974]. See §12.6.11.

`haloMassFunctionShethTormen` The halo mass function is computed from the function given by Sheth et al. [2001]. See §12.6.11.

`haloMassFunctionSimpleSystematic` The halo mass function is computed by modifying another halo mass function using a simple model for systematic errors. See §12.6.11.

`haloMassFunctionEnvironmentAveraged` The halo mass function is computed averaging another (presumably environment-dependent) mass function over environment.

`haloMassFunctionTinker2008Form` The halo mass function is computed from the function given by Tinker et al. [2008].

`haloMassFunctionTinker2008Generic` The halo mass function is computed from the function given by Tinker et al. [2008] with user-specified parameters.

Halo Model Power Spectrum Modifier

Additional implementations for halo model power spectrum modifier are added using the `haloModelPowerSpectrumModifier` class. The implementation should be placed in a file containing the directive:

```
!# <haloModelPowerSpectrumModifier name="haloModelPowerSpectrumModifierMyImplementation">
!# <description>A short description of the implementation.</description>
!# </haloModelPowerSpectrumModifier>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `haloModelPowerSpectrumModifierMyImplementation`. The file *must* define a type that extends the `haloModelPowerSpectrumModifierClass` class (or extends another type which is itself an extension of the `haloModelPowerSpectrumModifierClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (haloModelPowerSpectrumModifierClass), intent(inout)           :: self
  logical                                     , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (haloModelPowerSpectrumModifierClass), intent(inout)           :: self
 type   (inputParameters                    ), intent(inout)           :: descriptor
 logical                                     , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(haloModelPowerSpectrumModifierClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (haloModelPowerSpectrumModifierClass), intent(inout) :: self
 integer                                     , intent(in   ) :: stateFile
 type   (fgsl_file                           ), intent(in   ) :: fgslStateFile
 integer(c_size_t                            ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (haloModelPowerSpectrumModifierClass)                          , intent(&
&  inout) :: self
 type     (varying_string                     ), allocatable, dimension(:), intent(&
&  inout) :: allowedParameters
 character(len=*                              )                          , intent(in  &
&   ) :: sourceName
 end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(haloModelPowerSpectrumModifierClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(haloModelPowerSpectrumModifierClass), intent(inout) :: self
 class(haloModelPowerSpectrumModifierClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`modify` Modify the power spectra in the halo model of clustering. Must have the following interface:

```
subroutine myImplementationModify(self,wavenumber,term,powerSpectrum,&
&  powerSpectrumCovariance,mass)
 class           (haloModelPowerSpectrumModifierClass)               , intent(inout)  &
&           :: self
 double precision                                     , dimension(:)  , intent(in   )  &
&           :: wavenumber
 integer                                                             , intent(in   )  &
&           :: term
 double precision                                     , dimension(:)  , intent(inout)  &
&           :: powerSpectrum
 double precision                                     , dimension(:,:), intent(inout), &
&  optional :: powerSpectrumCovariance
 double precision                                                    , intent(in   ), &
&  optional :: mass
 end subroutine myImplementationModify
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (haloModelPowerSpectrumModifierClass), intent(inout) :: self
 integer                                     , intent(in   ) :: stateFile
 type   (fgsl_file                           ), intent(in   ) :: fgslStateFile
 integer(c_size_t                            ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`haloModelPowerSpectrumModifierIdentity` A identity modifier for power spectra in the halo model of clustering.

`haloModelPowerSpectrumModifierTriaxiality` A triaxiality modifier for power spectra in the halo model of clustering based on the results of Smith and Watts [2005]. See §12.20.2.

Dark Matter Halo Spin Parameter Distributions

Additional implementations for dark matter halo spin parameter distributions are added using the `haloSpinDistribution` class. The implementation should be placed in a file containing the directive:

```
!# <haloSpinDistribution name="haloSpinDistributionMyImplementation">
!# <description>A short description of the implementation.</description>
!# </haloSpinDistribution>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `haloSpinDistribution-MyImplementation`. The file *must* define a type that extends the `haloSpinDistributionClass` class (or extends another type which is itself an extension of the `haloSpinDistributionClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
 class  (haloSpinDistributionClass), intent(inout)          :: self
 logical                           , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (haloSpinDistributionClass), intent(inout)          :: self
 type   (inputParameters          ), intent(inout)          :: descriptor
 logical                           , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(haloSpinDistributionClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (haloSpinDistributionClass), intent(inout) :: self
 integer                           , intent(in   ) :: stateFile
 type   (fgsl_file                ), intent(in   ) :: fgslStateFile
 integer(c_size_t                 ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (haloSpinDistributionClass)                          , intent(inout) :: self
 type     (varying_string           ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                    )                          , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`distribution` Return the spin distribution function, $p(\lambda)$, for the given `node`. It is assumed that `node` provides the value of the spin at which the distribution function should be evaluated. Must have the following interface:

```
double precision function myImplementationDistribution(self,node)
 class(haloSpinDistributionClass), intent(inout) :: self
 type (treeNode                 ), intent(inout) :: node
end double precision function myImplementationDistribution
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(haloSpinDistributionClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(haloSpinDistributionClass), intent(inout) :: self
 class(haloSpinDistributionClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`sample` Samples a spin parameter from the distribution for the given `node`. Must have the following interface:

```
double precision function myImplementationSample(self,node)
 class(haloSpinDistributionClass), intent(inout) :: self
 type (treeNode                 ), intent(inout) :: node
end double precision function myImplementationSample
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (haloSpinDistributionClass), intent(inout) :: self
 integer                           , intent(in   ) :: stateFile
 type   (fgsl_file                ), intent(in   ) :: fgslStateFile
 integer(c_size_t                 ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`haloSpinDistributionBett2007` A halo spin distribution using the fitting formula of Bett et al. [2007]. See §12.11.7.

`haloSpinDistributionNbodyErrors` A halo spin distribution which modifies another spin distribution to account for the effects of particle noise errors in spins measured in N-body simulations.

`haloSpinDistributionLogNormal` A log-normal halo spin distribution. See §12.11.7.

`haloSpinDistributionDeltaFunction` A δ-function halo spin distribution. See §12.11.7.

Cold Mode Hot Halo Mass Distributions Core Radii

Additional implementations for cold mode hot halo mass distributions core radii are added using the `hotHaloColdModeCoreRadii` class. The implementation should be placed in a file containing the directive:

```
!# <hotHaloColdModeCoreRadii name="hotHaloColdModeCoreRadiiMyImplementation">
!# <description>A short description of the implementation.</description>
!# </hotHaloColdModeCoreRadii>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `hotHaloColdModeCoreRadiiMyImplementation`. The file *must* define a type that extends the `hotHaloColdModeCoreRadiiClass` class (or extends another type which is itself an extension of the `hotHaloColdModeCoreRadiiClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
 class  (hotHaloColdModeCoreRadiiClass), intent(inout)           :: self
 logical                               , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (hotHaloColdModeCoreRadiiClass), intent(inout)           :: self
 type   (inputParameters              ), intent(inout)           :: descriptor
 logical                               , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(hotHaloColdModeCoreRadiiClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`radius` Return the core radius of the hot halo mass distribution. Must have the following interface:

```
double precision function myImplementationRadius(self,node)
 class(hotHaloColdModeCoreRadiiClass), intent(inout) :: self
 type (treeNode                     ), intent(inout) :: node
end double precision function myImplementationRadius
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (hotHaloColdModeCoreRadiiClass), intent(inout) :: self
 integer                               , intent(in   ) :: stateFile
 type   (fgsl_file                     ), intent(in   ) :: fgslStateFile
 integer(c_size_t                      ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (hotHaloColdModeCoreRadiiClass)                          , intent(inout) :: &
&  self
 type     (varying_string              ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                       )                          , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(hotHaloColdModeCoreRadiiClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(hotHaloColdModeCoreRadiiClass), intent(inout) :: self
 class(hotHaloColdModeCoreRadiiClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (hotHaloColdModeCoreRadiiClass), intent(inout) :: self
 integer                               , intent(in   ) :: stateFile
 type   (fgsl_file                    ), intent(in   ) :: fgslStateFile
 integer(c_size_t                     ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`hotHaloColdModeCoreRadiiVirialFraction` Provides an implementation of the cold mode hot halo mass distribution core radius class which sets the core radius to a fraction of the virial radius.

Hot Halo Mass Distributions

Additional implementations for hot halo mass distributions are added using the `hotHaloMassDistribution` class. The implementation should be placed in a file containing the directive:

```
!# <hotHaloMassDistribution name="hotHaloMassDistributionMyImplementation">
!# <description>A short description of the implementation.</description>
!# </hotHaloMassDistribution>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `hotHaloMassDistributionMyImplementation`. The file *must* define a type that extends the `hotHaloMassDistributionClass` class (or extends another type which is itself an extension of the `hotHaloMassDistributionClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
 class  (hotHaloMassDistributionClass), intent(inout)           :: self
 logical                              , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (hotHaloMassDistributionClass), intent(inout)           :: self
 type   (inputParameters             ), intent(inout)           :: descriptor
 logical                              , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(hotHaloMassDistributionClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`enclosedMass` Return the mass enclosed in the hot halo at the given `radius`. Must have the following interface:

```
double precision function myImplementationEnclosedMass(self,node,radius)
 class           (hotHaloMassDistributionClass), intent(inout)         :: self
 type            (treeNode                   ), intent(inout), target :: node
 double precision                             , intent(in   )         :: radius
end double precision function myImplementationEnclosedMass
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (hotHaloMassDistributionClass), intent(inout) :: self
 integer                              , intent(in   ) :: stateFile
 type   (fgsl_file                    ), intent(in   ) :: fgslStateFile
 integer(c_size_t                     ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (hotHaloMassDistributionClass)                          , intent(inout) :: &
&  self
 type     (varying_string              ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                       )                          , intent(in   ) :: &
&  sourceName
end subroutine myImplementationAllowedParameters
```

`density` Return the density of the hot halo at the given `radius`. Must have the following interface:

```
double precision function myImplementationDensity(self,node,radius)
 class           (hotHaloMassDistributionClass), intent(inout) :: self
 type            (treeNode                   ), intent(inout) :: node
 double precision                             , intent(in   ) :: radius
end double precision function myImplementationDensity
```

`rotationNormalization` Returns the relation between specific angular momentum and rotation velocity (assuming a rotation velocity that is constant in radius) for `node`. Specifically, the normalization, A, returned is such that $V_{\rm rot} = AJ/M$. Must have the following interface:

```
double precision function myImplementationRotationNormalization(self,node)
 class(hotHaloMassDistributionClass), intent(inout) :: self
 type (treeNode                   ), intent(inout) :: node
end double precision function myImplementationRotationNormalization
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(hotHaloMassDistributionClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(hotHaloMassDistributionClass), intent(inout) :: self
 class(hotHaloMassDistributionClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (hotHaloMassDistributionClass), intent(inout) :: self
 integer                              , intent(in   ) :: stateFile
 type   (fgsl_file                    ), intent(in   ) :: fgslStateFile
 integer(c_size_t                     ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

`densityLogSlope` Return the logarithmic slope of the density of the hot halo at the given `radius`. Must have the following interface:

```
double precision function myImplementationDensityLogSlope(self,node,radius)
 class           (hotHaloMassDistributionClass), intent(inout) :: self
 type            (treeNode                    ), intent(inout) :: node
 double precision                              , intent(in   ) :: radius
end double precision function myImplementationDensityLogSlope
```

`radialMoment` Return the density of the hot halo at the given `radius`. Must have the following interface:

```
double precision function myImplementationRadialMoment(self,node,moment, radius)
 class           (hotHaloMassDistributionClass), intent(inout) :: self
 type            (treeNode                    ), intent(inout) :: node
 double precision                              , intent(in   ) :: moment, radius
end double precision function myImplementationRadialMoment
```

Existing implementations are:

`hotHaloMassDistributionBetaProfile` Provides a β-profile implementation of the hot halo mass distribution class. See §12.21.1.

`hotHaloMassDistributionRicotti2000` Provides an implementation of the hot halo mass distribution class which uses the model of Ricotti and Shull [2000]. See §12.21.2.

`hotHaloMassDistributionEnzoHydrostatic` Provides an implementation of the hot halo mass distribution class which uses the "hydrostatic" profile used by the Enzo simulation code. See §12.21.4.

`hotHaloMassDistributionNull` Provides a null implementation of the hot halo mass distribution class. See §12.21.3.

`hotHaloMassDistributionPatejLoeb2015` Provides an implementation of the hot halo mass distribution class which uses the model of Patej and Loeb [2015].

Hot Halo Mass Distributions Core Radii

Additional implementations for hot halo mass distributions core radii are added using the `hotHaloMassDistributionCoreRadi` class. The implementation should be placed in a file containing the directive:

```
!# <hotHaloMassDistributionCoreRadius name="hotHaloMassDistributionCoreRadiusMyImplementation">
!# <description>A short description of the implementation.</description>
!# </hotHaloMassDistributionCoreRadius>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `hotHaloMassDistributionCoreRadiusMyImplementation`. The file *must* define a type that extends the `hotHaloMassDistributionCoreRadiusClass` class (or extends another type which is itself an extension of the `hotHaloMassDistributionCoreRadiusClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (hotHaloMassDistributionCoreRadiusClass), intent(inout)           :: self
  logical                                        , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationDescriptor(self,descriptor,includeMethod)
  class  (hotHaloMassDistributionCoreRadiusClass), intent(inout)           :: self
  type   (inputParameters                       ), intent(inout)           :: descriptor
  logical                                        , intent(in   ), optional :: &
&  includeMethod
 end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAutoHook(self)
  class(hotHaloMassDistributionCoreRadiusClass), intent(inout) :: self
 end subroutine myImplementationAutoHook
```

`radius` Return the core radius of the hot halo mass distribution. Must have the following interface:

```
 double precision function myImplementationRadius(self,node)
  class(hotHaloMassDistributionCoreRadiusClass), intent(inout) :: self
  type (treeNode                               ), intent(inout) :: node
 end double precision function myImplementationRadius
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (hotHaloMassDistributionCoreRadiusClass), intent(inout) :: self
 integer                                        , intent(in   ) :: stateFile
 type   (fgsl_file                              ), intent(in   ) :: fgslStateFile
 integer(c_size_t                               ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (hotHaloMassDistributionCoreRadiusClass)                          , intent(&
&  inout) :: self
 type     (varying_string                        ), allocatable, dimension(:), intent(&
&  inout) :: allowedParameters
 character(len=*                                 )                          , intent(&
&  in   ) :: sourceName
 end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(hotHaloMassDistributionCoreRadiusClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(hotHaloMassDistributionCoreRadiusClass), intent(inout) :: self
 class(hotHaloMassDistributionCoreRadiusClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (hotHaloMassDistributionCoreRadiusClass), intent(inout) :: self
 integer                                        , intent(in   ) :: stateFile
 type   (fgsl_file                              ), intent(in   ) :: fgslStateFile
 integer(c_size_t                               ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`hotHaloMassDistributionCoreRadiusVirialFraction` Provides an implementation of the hot halo mass distribution core radius class which sets the core radius to a fraction of the virial radius. See §12.22.2.

`hotHaloMassDistributionCoreRadiusGrowing` Provides an implementation of the hot halo mass distribution core radius class in which the core grows as the hot halo content is depleted. See §12.22.1.

Hot Halo Outflow Reincorporation

Additional implementations for hot halo outflow reincorporation are added using the `hotHaloOutflowReincorporation` class. The implementation should be placed in a file containing the directive:

```
!# <hotHaloOutflowReincorporation name="hotHaloOutflowReincorporationMyImplementation">
!# <description>A short description of the implementation.</description>
!# </hotHaloOutflowReincorporation>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `hotHaloOutflowReincorporationMyImplementation`. The file *must* define a type that extends the `hotHaloOutflowReincorporationClass` class (or extends another type which is itself an extension of the `hotHaloOutflowReincorporationClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (hotHaloOutflowReincorporationClass), intent(inout)          :: self
  logical                                    , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (hotHaloOutflowReincorporationClass), intent(inout)          :: self
 type   (inputParameters                   ), intent(inout)          :: descriptor
 logical                                    , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(hotHaloOutflowReincorporationClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (hotHaloOutflowReincorporationClass), intent(inout) :: self
 integer                                    , intent(in   ) :: stateFile
 type   (fgsl_file                         ), intent(in   ) :: fgslStateFile
 integer(c_size_t                          ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (hotHaloOutflowReincorporationClass)                          , intent(inout&
&  ) :: self
 type     (varying_string                    ), allocatable, dimension(:), intent(inout&
&  ) :: allowedParameters
 character(len=*                             )                          , intent(in   &
&  ) :: sourceName
end subroutine myImplementationAllowedParameters
```

`rate` Return the rate at which outflowed mass is being reincorporated into the hot halo. Must have the following interface:

```
double precision function myImplementationRate(self,node)
 class(hotHaloOutflowReincorporationClass), intent(inout) :: self
 type (treeNode                          ), intent(inout) :: node
end double precision function myImplementationRate
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(hotHaloOutflowReincorporationClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(hotHaloOutflowReincorporationClass), intent(inout) :: self
 class(hotHaloOutflowReincorporationClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (hotHaloOutflowReincorporationClass), intent(inout) :: self
 integer                                    , intent(in   ) :: stateFile
 type   (fgsl_file                          ), intent(in   ) :: fgslStateFile
 integer(c_size_t                           ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`hotHaloOutflowReincorporationHaloDynamicalTime` An implementation of the hot halo outflow reincorporation class in which reincorporation occurs on a multiple of the halo dynamical timescale.

`hotHaloOutflowReincorporationZero` An implementation of the hot halo outflow reincorporation class which gives zero reincorporation rate.

`hotHaloOutflowReincorporationVelocityMaximumScaling` An implementation of the hot halo outflow reincorporation class which uses simple scalings based on the halo maximum circular velocity. See §12.26.1.

`hotHaloOutflowReincorporationHenriques2013` An implementation of the hot halo outflow reincorporation class which implements the model of Henriques et al. [2013]. Specifically, outflowed gas is returned at a rate:

$$\dot{M}_{\rm return} = \gamma(1+z)^{-\delta_1}\left(\frac{V_{\rm vir}}{200{\rm km/s}}\right)^{\delta_2}\frac{M_{\rm outflowed}}{\tau_{\rm dyn}} \tag{16.8}$$

Models of ram pressure force from the hot halo.

Additional implementations for models of ram pressure force from the hot halo. are added using the `hotHaloRamPressureForce` class. The implementation should be placed in a file containing the directive:

```
!# <hotHaloRamPressureForce name="hotHaloRamPressureForceMyImplementation">
!# <description>A short description of the implementation.</description>
!# </hotHaloRamPressureForce>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `hotHaloRamPressureForceMyImplementation`. The file *must* define a type that extends the `hotHaloRamPressureForceClass` class (or extends another type which is itself an extension of the `hotHaloRamPressureForceClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (hotHaloRamPressureForceClass), intent(inout)           :: self
  logical                              , intent(in   ), optional :: includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationDescriptor(self,descriptor,includeMethod)
  class  (hotHaloRamPressureForceClass), intent(inout)           :: self
  type   (inputParameters             ), intent(inout)           :: descriptor
  logical                              , intent(in   ), optional :: includeMethod
 end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAutoHook(self)
  class(hotHaloRamPressureForceClass), intent(inout) :: self
 end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (hotHaloRamPressureForceClass), intent(inout) :: self
 integer                              , intent(in   ) :: stateFile
 type   (fgsl_file                    ), intent(in   ) :: fgslStateFile
 integer(c_size_t                     ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (hotHaloRamPressureForceClass)                          , intent(inout) :: &
&  self
 type     (varying_string             ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                      )                          , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`force` Return the ram pressure force acting on `node` (in units of $(\mathrm{km/s})^2 M_\odot/\mathrm{Mpc}^3$). Must have the following interface:

```
double precision function myImplementationForce(self,node)
 class(hotHaloRamPressureForceClass), intent(inout) :: self
 type (treeNode                    ), intent(inout) :: node
end double precision function myImplementationForce
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(hotHaloRamPressureForceClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(hotHaloRamPressureForceClass), intent(inout) :: self
 class(hotHaloRamPressureForceClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (hotHaloRamPressureForceClass), intent(inout) :: self
 integer                              , intent(in   ) :: stateFile
 type   (fgsl_file                    ), intent(in   ) :: fgslStateFile
 integer(c_size_t                     ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`hotHaloRamPressureForceZero` A hot halo ram pressure force class which follows the model of Font et al. [2008].

`hotHaloRamPressureForceFont2008` A hot halo ram pressure force class which follows the model of Font et al. [2008].

Models of ram pressure stripping due to the hot halo.

Additional implementations for models of ram pressure stripping due to the hot halo. are added using the `hotHaloRamPressureStripping` class. The implementation should be placed in a file containing the directive:

```
!# <hotHaloRamPressureStripping name="hotHaloRamPressureStrippingMyImplementation">
!# <description>A short description of the implementation.</description>
!# </hotHaloRamPressureStripping>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `hotHaloRamPressureStrippingMyImplementation`. The file *must* define a type that extends the `hotHaloRamPressureStrippingClass` class (or extends another type which is itself an extension of the `hotHaloRamPressureStrippingClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (hotHaloRamPressureStrippingClass), intent(inout)          :: self
  logical                                  , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (hotHaloRamPressureStrippingClass), intent(inout)          :: self
 type   (inputParameters                 ), intent(inout)          :: descriptor
 logical                                  , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(hotHaloRamPressureStrippingClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (hotHaloRamPressureStrippingClass), intent(inout) :: self
 integer                                  , intent(in   ) :: stateFile
 type   (fgsl_file                        ), intent(in   ) :: fgslStateFile
 integer(c_size_t                         ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (hotHaloRamPressureStrippingClass)                          , intent(inout) &
&  :: self
 type     (varying_string                 ), allocatable, dimension(:), intent(inout) &
&  :: allowedParameters
 character(len=*                          )                          , intent(in   ) &
&  :: sourceName
 end subroutine myImplementationAllowedParameters
```

`radiusStripped` Return the radius to which `node` is stripped due to ram pressure from its host halo (in units of Mpc). Must have the following interface:

```
double precision function myImplementationRadiusStripped(self,node)
 class(hotHaloRamPressureStrippingClass), intent(inout)         :: self
 type (treeNode                        ), intent(inout), target :: node
end double precision function myImplementationRadiusStripped
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(hotHaloRamPressureStrippingClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(hotHaloRamPressureStrippingClass), intent(inout) :: self
 class(hotHaloRamPressureStrippingClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (hotHaloRamPressureStrippingClass), intent(inout) :: self
 integer                                  , intent(in   ) :: stateFile
 type   (fgsl_file                        ), intent(in   ) :: fgslStateFile
 integer(c_size_t                         ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`hotHaloRamPressureStrippingVirialRadius` A hot halo ram pressure stripping class which simply returns the virial radius.

`hotHaloRamPressureStrippingFont2008` A hot halo ram pressure stripping class based on the methods of Font et al. [2008].

Models of ram pressure stripping timescales due to the hot halo.

Additional implementations for models of ram pressure stripping timescales due to the hot halo. are added using the `hotHaloRamPressureTimescale` class. The implementation should be placed in a file containing the directive:

```
!# <hotHaloRamPressureTimescale name="hotHaloRamPressureTimescaleMyImplementation">
!# <description>A short description of the implementation.</description>
!# </hotHaloRamPressureTimescale>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `hotHaloRamPressure-TimescaleMyImplementation`. The file *must* define a type that extends the `hotHaloRamPressureTimescaleClass` class (or extends another type which is itself an extension of the `hotHaloRamPressureTimescaleClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (hotHaloRamPressureTimescaleClass), intent(inout)           :: self
  logical                                  , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (hotHaloRamPressureTimescaleClass), intent(inout)           :: self
 type   (inputParameters                 ), intent(inout)           :: descriptor
 logical                                  , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(hotHaloRamPressureTimescaleClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (hotHaloRamPressureTimescaleClass), intent(inout) :: self
 integer                                  , intent(in   ) :: stateFile
 type   (fgsl_file                       ), intent(in   ) :: fgslStateFile
 integer(c_size_t                        ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (hotHaloRamPressureTimescaleClass)                          , intent(inout) &
&  :: self
 type     (varying_string                 ), allocatable, dimension(:), intent(inout) &
&  :: allowedParameters
 character(len=*                          )                          , intent(in   ) &
&  :: sourceName
 end subroutine myImplementationAllowedParameters
```

`timescale` Return the ram pressure stripping timescale for `node` (in units of Gyr). Must have the following interface:

```
double precision function myImplementationTimescale(self,node)
 class(hotHaloRamPressureTimescaleClass), intent(inout) :: self
 type (treeNode                        ), intent(inout) :: node
end double precision function myImplementationTimescale
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(hotHaloRamPressureTimescaleClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(hotHaloRamPressureTimescaleClass), intent(inout) :: self
 class(hotHaloRamPressureTimescaleClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (hotHaloRamPressureTimescaleClass), intent(inout) :: self
 integer                                  , intent(in   ) :: stateFile
 type   (fgsl_file                       ), intent(in   ) :: fgslStateFile
 integer(c_size_t                        ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`hotHaloRamPressureTimescaleHaloDynamicalTime` A hot halo ram pressure timescale class in which the timescale is equal to the halo dynamical time.

`hotHaloRamPressureTimescaleRamPressureAcceleration` A hot halo ram pressure timescale class in which the timescale is estimated from the ram pressure acceleration.

Hot halo temperature profiles

Additional implementations for hot halo temperature profiles are added using the `hotHaloTemperatureProfile` class. The implementation should be placed in a file containing the directive:

```
!# <hotHaloTemperatureProfile name="hotHaloTemperatureProfileMyImplementation">
!# <description>A short description of the implementation.</description>
!# </hotHaloTemperatureProfile>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `hotHaloTemperatureProfileMyImplementation`. The file *must* define a type that extends the `hotHaloTemperatureProfileClass` class (or extends another type which is itself an extension of the `hotHaloTemperatureProfileClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`temperatureLogSlope` Return the logarithmic slope of the temperature of the hot halo at the given `radius`. Must have the following interface:

```
double precision function myImplementationTemperatureLogSlope(self,node,radius)
 class           (hotHaloTemperatureProfileClass), intent(inout) :: self
 type            (treeNode                      ), intent(inout) :: node
 double precision                                , intent(in   ) :: radius
end double precision function myImplementationTemperatureLogSlope
```

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (hotHaloTemperatureProfileClass), intent(inout)           :: self
  logical                                , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (hotHaloTemperatureProfileClass), intent(inout)           :: self
 type   (inputParameters               ), intent(inout)           :: descriptor
 logical                                , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(hotHaloTemperatureProfileClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (hotHaloTemperatureProfileClass), intent(inout) :: self
 integer                                , intent(in   ) :: stateFile
 type   (fgsl_file                      ), intent(in   ) :: fgslStateFile
 integer(c_size_t                       ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (hotHaloTemperatureProfileClass)                          , intent(inout) ::&
&  self
 type     (varying_string               ), allocatable, dimension(:), intent(inout) ::&
&  allowedParameters
 character(len=*                         )                          , intent(in   ) ::&
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(hotHaloTemperatureProfileClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(hotHaloTemperatureProfileClass), intent(inout) :: self
 class(hotHaloTemperatureProfileClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (hotHaloTemperatureProfileClass), intent(inout) :: self
 integer                                , intent(in   ) :: stateFile
 type   (fgsl_file                      ), intent(in   ) :: fgslStateFile
 integer(c_size_t                       ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

`temperature` Return the temperature of the hot halo at the given `radius`. Must have the following interface:

```
double precision function myImplementationTemperature(self,node,radius)
 class           (hotHaloTemperatureProfileClass), intent(inout) :: self
 type            (treeNode                      ), intent(inout) :: node
 double precision                                , intent(in   ) :: radius
end double precision function myImplementationTemperature
```

Existing implementations are:

`hotHaloTemperatureProfileVirial` Provides an implementation of the hot halo temperature profile class which uses an isothermal virial temperature. See §12.27.1.

`hotHaloTemperatureProfileEnzoHydrostatic` Provides an implementation of the hot halo temperature profile class which uses the "hydrostatic" solution from the Enzo code. See §12.27.2.

Initial Mass Functions

Additional implementations for initial mass functions are added using the `initialMassFunction` class. The implementation should be placed in a file containing the directive:

```
!# <initialMassFunction name="initialMassFunctionMyImplementation">
!# <description>A short description of the implementation.</description>
!# </initialMassFunction>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `initialMassFunction-MyImplementation`. The file *must* define a type that extends the `initialMassFunctionClass` class (or extends another type which is itself an extension of the `initialMassFunctionClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (initialMassFunctionClass), intent(inout)          :: self
  logical                          , intent(in   ), optional :: includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationDescriptor(self,descriptor,includeMethod)
  class  (initialMassFunctionClass), intent(inout)          :: self
  type   (inputParameters         ), intent(inout)          :: descriptor
  logical                          , intent(in   ), optional :: includeMethod
 end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(initialMassFunctionClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (initialMassFunctionClass), intent(inout) :: self
 integer                          , intent(in   ) :: stateFile
 type   (fgsl_file               ), intent(in   ) :: fgslStateFile
 integer(c_size_t                ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`tabulate` Return the initial mass function, $\phi(M) = \mathrm{d}N/\mathrm{d}M$, at the given mass $M =$`initialMass`. Must have the following interface:

```
subroutine myImplementationTabulate(self,imfTable)
 class(initialMassFunctionClass)             , intent(inout) :: self
 class(table1D                 ), allocatable, intent(inout) :: imfTable
end subroutine myImplementationTabulate
```

`massMinimum` Return the minimum mass in the initial mass function. Must have the following interface:

```
double precision function myImplementationMassMinimum(self)
 class(initialMassFunctionClass), intent(inout) :: self
end double precision function myImplementationMassMinimum
```

`massMaximum` Return the maximum mass in the initial mass function. Must have the following interface:

```
double precision function myImplementationMassMaximum(self)
 class(initialMassFunctionClass), intent(inout) :: self
end double precision function myImplementationMassMaximum
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (initialMassFunctionClass)                          , intent(inout) :: self
 type     (varying_string          ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                   )                          , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(initialMassFunctionClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`phi` Return the initial mass function, $\phi(M) = \mathrm{d}N/\mathrm{d}M$, at the given mass $M =$`massInitial`. Must have the following interface:

```
double precision function myImplementationPhi(self,massInitial)
 class           (initialMassFunctionClass), intent(inout) :: self
 double precision                          , intent(in   ) :: massInitial
end double precision function myImplementationPhi
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(initialMassFunctionClass), intent(inout) :: self
 class(initialMassFunctionClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`label` Return the label for this IMF. Must have the following interface:

```
type(varying_string) function myImplementationLabel(self)
 class(initialMassFunctionClass), intent(inout) :: self
end type(varying_string) function myImplementationLabel
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (initialMassFunctionClass), intent(inout) :: self
 integer                          , intent(in   ) :: stateFile
 type   (fgsl_file               ), intent(in   ) :: fgslStateFile
 integer(c_size_t                ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`initialMassFunctionPiecewisePowerLaw` A stellar initial mass function class for piecewise power-law IMFs.

`initialMassFunctionChabrier2001` A stellar initial mass function class based on Chabrier [2001].

`initialMassFunctionSalpeter1955` A stellar initial mass function class for the Salpeter [1955] IMF.

`initialMassFunctionKennicutt1983` A stellar initial mass function class for the Kennicutt [1983] IMF.

`initialMassFunctionScalo1986` A stellar initial mass function class for the Scalo [1986] IMF.

`initialMassFunctionBPASS` A stellar initial mass function class used by the BPASS library.

`initialMassFunctionBaugh2005TopHeavy` A stellar initial mass function class for the top-heavy stellar initial mass function from Baugh et al. [2005].

`initialMassFunctionMillerScalo1979` A stellar initial mass function class for the Miller and Scalo [1979] IMF.

`initialMassFunctionKroupa2001` A stellar initial mass function class for the Kroupa [2001] IMF.

Intergalactic Medium Filtering Mass

Additional implementations for intergalactic medium filtering mass are added using the `intergalacticMediumFilteringMass` class. The implementation should be placed in a file containing the directive:

```
!# <intergalacticMediumFilteringMass name="intergalacticMediumFilteringMassMyImplementation">
!# <description>A short description of the implementation.</description>
!# </intergalacticMediumFilteringMass>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `intergalacticMediumFilteringMassMyImplementation`. The file *must* define a type that extends the `intergalacticMediumFilteringMassClass` class (or extends another type which is itself an extension of the `intergalacticMediumFilteringMassClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (intergalacticMediumFilteringMassClass), intent(inout)           :: self
  logical                                       , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationDescriptor(self,descriptor,includeMethod)
  class  (intergalacticMediumFilteringMassClass), intent(inout)           :: self
  type   (inputParameters                      ), intent(inout)           :: descriptor
  logical                                       , intent(in   ), optional :: &
&  includeMethod
 end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAutoHook(self)
  class(intergalacticMediumFilteringMassClass), intent(inout) :: self
 end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
  class  (intergalacticMediumFilteringMassClass), intent(inout) :: self
  integer                                       , intent(in   ) :: stateFile
  type   (fgsl_file                             ), intent(in   ) :: fgslStateFile
  integer(c_size_t                              ), intent(in   ) :: stateOperationID
 end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (intergalacticMediumFilteringMassClass)                          , intent(&
&  inout) :: self
 type     (varying_string                      ), allocatable, dimension(:), intent(&
&  inout) :: allowedParameters
 character(len=*                                )                          , intent(in&
&     ) :: sourceName
end subroutine myImplementationAllowedParameters
```

`massFiltering` Return the filtering mass at the given `time`. Must have the following interface:

```
double precision function myImplementationMassFiltering(self,time)
 class           (intergalacticMediumFilteringMassClass), intent(inout) :: self
 double precision                                       , intent(in   ) :: time
end double precision function myImplementationMassFiltering
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(intergalacticMediumFilteringMassClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(intergalacticMediumFilteringMassClass), intent(inout) :: self
 class(intergalacticMediumFilteringMassClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (intergalacticMediumFilteringMassClass), intent(inout) :: self
 integer                                       , intent(in   ) :: stateFile
 type   (fgsl_file                             ), intent(in   ) :: fgslStateFile
 integer(c_size_t                              ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`intergalacticMediumFilteringMassGnedin2000` An implementation of the Gnedin [2000] filtering mass calculation.

Intergalactic Medium State

Additional implementations for intergalactic medium state are added using the `intergalacticMediumState` class. The implementation should be placed in a file containing the directive:

```
!# <intergalacticMediumState name="intergalacticMediumStateMyImplementation">
!# <description>A short description of the implementation.</description>
!# </intergalacticMediumState>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `intergalacticMediumStateMyImplementation`. The file *must* define a type that extends the `intergalacticMediumStateClass` class (or extends another type which is itself an extension of the `intergalacticMediumStateClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`singlyIonizedHeliumFraction` Return the singly-ionized fraction of helium in the IGM at the given time. Must have the following interface:

```
double precision function myImplementationSinglyIonizedHeliumFraction(self,time)
 class           (intergalacticMediumStateClass), intent(inout) :: self
 double precision                               , intent(in   ) :: time
end double precision function myImplementationSinglyIonizedHeliumFraction
```

`massJeans` Return the instantaneus Jeans mass (in $M_\odot$) at the given time. A default implementation exists. If overridden the following interface must be used:

```
double precision function myImplementationMassJeans(self,time)
 class           (intergalacticMediumStateClass), intent(inout) :: self
 double precision                               , intent(in   ) :: time
end double precision function myImplementationMassJeans
```

`electronScatteringTime` Return the cosmological time at which the given electron scattering `opticalDepth` is reached (integrating from the present day) in the IGM. A default implementation exists. If overridden the following interface must be used:

```
double precision function myImplementationElectronScatteringTime(self,opticalDepth,&
&  assumeFullyIonized)
  class           (intergalacticMediumStateClass), intent(inout)           :: self
  double precision                               , intent(in   )           :: &
&  opticalDepth
  logical                                        , intent(in   ), optional :: &
&  assumeFullyIonized
 end double precision function myImplementationElectronScatteringTime
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(intergalacticMediumStateClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`electronScatteringOpticalDepth` Return the electron scattering optical depth from the present day back to the given `time` in the IGM. A default implementation exists. If overridden the following interface must be used:

```
 double precision function myImplementationElectronScatteringOpticalDepth(self,time,&
&  assumeFullyIonized)
  class           (intergalacticMediumStateClass), intent(inout)           :: self
  double precision                               , intent(in   )           :: time
  logical                                        , intent(in   ), optional :: &
&  assumeFullyIonized
 end double precision function myImplementationElectronScatteringOpticalDepth
```

`neutralHydrogenFraction` Return the neutral fraction of hydrogen in the IGM at the given time. Must have the following interface:

```
double precision function myImplementationNeutralHydrogenFraction(self,time)
 class           (intergalacticMediumStateClass), intent(inout) :: self
 double precision                               , intent(in   ) :: time
end double precision function myImplementationNeutralHydrogenFraction
```

`temperature` Return the temperature (in Kelvin) of the IGM at the given time. Must have the following interface:

```
double precision function myImplementationTemperature(self,time)
 class           (intergalacticMediumStateClass), intent(inout) :: self
 double precision                               , intent(in   ) :: time
end double precision function myImplementationTemperature
```

`doublyIonizedHeliumFraction` Return the doubly-ionized fraction of helium in the IGM at the given time. A default implementation exists. If overridden the following interface must be used:

```
double precision function myImplementationDoublyIonizedHeliumFraction(self,time)
 class           (intergalacticMediumStateClass), intent(inout) :: self
 double precision                               , intent(in   ) :: time
end double precision function myImplementationDoublyIonizedHeliumFraction
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (intergalacticMediumStateClass), intent(inout)           :: self
 type   (inputParameters              ), intent(inout)           :: descriptor
 logical                               , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
 class  (intergalacticMediumStateClass), intent(inout)           :: self
 logical                               , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`electronFraction` Return the electron fraction (relative to hydrogen) in the IGM at the given time. Must have the following interface:

```
double precision function myImplementationElectronFraction(self,time)
 class           (intergalacticMediumStateClass), intent(inout) :: self
 double precision                               , intent(in   ) :: time
end double precision function myImplementationElectronFraction
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(intergalacticMediumStateClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (intergalacticMediumStateClass), intent(inout) :: self
 integer                               , intent(in   ) :: stateFile
 type   (fgsl_file                    ), intent(in   ) :: fgslStateFile
 integer(c_size_t                     ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (intergalacticMediumStateClass)                          , intent(inout) :: &
&  self
 type     (varying_string               ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                         )                          , intent(in   ) :: &
&  sourceName
end subroutine myImplementationAllowedParameters
```

`neutralHeliumFraction` Return the neutral fraction of helium in the IGM at the given time. Must have the following interface:

```
double precision function myImplementationNeutralHeliumFraction(self,time)
 class           (intergalacticMediumStateClass), intent(inout) :: self
 double precision                               , intent(in   ) :: time
end double precision function myImplementationNeutralHeliumFraction
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(intergalacticMediumStateClass), intent(inout) :: self
 class(intergalacticMediumStateClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (intergalacticMediumStateClass), intent(inout) :: self
 integer                               , intent(in   ) :: stateFile
 type   (fgsl_file                     ), intent(in   ) :: fgslStateFile
 integer(c_size_t                      ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

`singlyIonizedHydrogenFraction` Return the singly-ionized fraction of hydrogen in the IGM at the given time. A default implementation exists. If overridden the following interface must be used:

```
double precision function myImplementationSinglyIonizedHydrogenFraction(self,time)
 class           (intergalacticMediumStateClass), intent(inout) :: self
 double precision                               , intent(in   ) :: time
end double precision function myImplementationSinglyIonizedHydrogenFraction
```

Existing implementations are:

`intergalacticMediumStateInstantReionization` The intergalactic medium is assumed to be instantaneously and fully reionized at a fixed redshift, and heated to a fixed temperature. Prior to that, the reionization state is provided by some other class.

`intergalacticMediumStateSimple` The intergalactic medium is assumed to be instantaneously and fully reionized at a fixed redshift, and heated to a fixed temperature. See §12.29.1.

`intergalacticMediumStateInternal` The state of the intergalactic medium is solved for internally.

`intergalacticMediumStateFile` The intergalactic medium state is read from file. See §12.29.3.

`intergalacticMediumStateRecFast` The intergalactic medium state is computed using RecFast. See §12.29.2.

Linear Growth of Cosmological Structure

Additional implementations for linear growth of cosmological structure are added using the `linearGrowth` class. The implementation should be placed in a file containing the directive:

```
!# <linearGrowth name="linearGrowthMyImplementation">
!# <description>A short description of the implementation.</description>
!# </linearGrowth>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `linearGrowthMyImplementation`. The file *must* define a type that extends the `linearGrowthClass` class (or extends another type which is itself an extension of the `linearGrowthClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (linearGrowthClass), intent(inout)           :: self
  logical                   , intent(in   ), optional :: includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationDescriptor(self,descriptor,includeMethod)
  class  (linearGrowthClass), intent(inout)           :: self
  type   (inputParameters  ), intent(inout)           :: descriptor
  logical                   , intent(in   ), optional :: includeMethod
 end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAutoHook(self)
  class(linearGrowthClass), intent(inout) :: self
 end subroutine myImplementationAutoHook
```

`value` Return the linear growth factor at the given time and mass. Must have the following interface:

```
 double precision function myImplementationValue(self,time      , expansionFactor,&
&  collapsing,normalize , component,wavenumber)
  class           (linearGrowthClass), intent(inout)           :: self
  double precision                   , intent(in   ), optional :: time, expansionFactor
  logical                            , intent(in   ), optional :: collapsing
  integer                            , intent(in   ), optional :: normalize, component
  double precision                   , intent(in   ), optional :: wavenumber
 end double precision function myImplementationValue
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
  class  (linearGrowthClass), intent(inout) :: self
  integer                   , intent(in   ) :: stateFile
  type   (fgsl_file        ), intent(in   ) :: fgslStateFile
  integer(c_size_t         ), intent(in   ) :: stateOperationID
 end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
  class    (linearGrowthClass)                          , intent(inout) :: self
  type     (varying_string   ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
  character(len=*            )                          , intent(in   ) :: sourceName
 end subroutine myImplementationAllowedParameters
```

`logarithmicDerivativeExpansionFactor` Return the logarithmic derivative of linear growth factor with respect to expansion factor. Must have the following interface:

```
 double precision function myImplementationLogarithmicDerivativeExpansionFactor(self,&
&  time      , expansionFactor,collapsing,component,wavenumber)
  class           (linearGrowthClass), intent(inout)           :: self
  double precision                   , intent(in   ), optional :: time, expansionFactor
  logical                            , intent(in   ), optional :: collapsing
  integer                            , intent(in   ), optional :: component
  double precision                   , intent(in   ), optional :: wavenumber
 end double precision function myImplementationLogarithmicDerivativeExpansionFactor
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(linearGrowthClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(linearGrowthClass), intent(inout) :: self
 class(linearGrowthClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (linearGrowthClass), intent(inout) :: self
 integer                   , intent(in   ) :: stateFile
 type   (fgsl_file        ), intent(in   ) :: fgslStateFile
 integer(c_size_t         ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`linearGrowthSimple` Linear growth of cosmological structure in simple cosomologies. Ignores pressure terms for the growth of baryons and has no wavenumber dependence. Also assumes no growth of radiation perturbations.

Mass Distributions

Additional implementations for mass distributions are added using the `massDistribution` class. The implementation should be placed in a file containing the directive:

```
!# <massDistribution name="massDistributionMyImplementation">
!# <description>A short description of the implementation.</description>
!# </massDistribution>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `massDistribution-MyImplementation`. The file *must* define a type that extends the `massDistributionClass` class (or

extends another type which is itself an extension of the `massDistributionClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`potential` Return the gravitational potential of the distribution at the given coordinates. Must have the following interface:

```
double precision function myImplementationPotential(self,coordinates)
 class(massDistributionClass), intent(inout) :: self
 class(coordinate           ), intent(in   ) :: coordinates
end double precision function myImplementationPotential
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(massDistributionClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`massEnclosedBySphere` Return the mass enclosed in the distribution by a sphere of given radius. Must have the following interface:

```
double precision function myImplementationMassEnclosedBySphere(self,radius)
 class           (massDistributionClass), intent(inout) :: self
 double precision                       , intent(in   ) :: radius
end double precision function myImplementationMassEnclosedBySphere
```

`densityGradientRadial` Return the radial gradient of density of the distribution at the given coordinates. Must have the following interface:

```
double precision function myImplementationDensityGradientRadial(self,coordinates,&
&  logarithmic)
 class  (massDistributionClass), intent(inout)           :: self
 class  (coordinate           ), intent(in   )           :: coordinates
 logical                       , intent(in   ), optional :: logarithmic
end double precision function myImplementationDensityGradientRadial
```

`massTotal` Return the total mass of the distribution. Must have the following interface:

```
double precision function myImplementationMassTotal(self)
 class(massDistributionClass), intent(inout) :: self
end double precision function myImplementationMassTotal
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (massDistributionClass), intent(inout)           :: self
 type   (inputParameters      ), intent(inout)           :: descriptor
 logical                       , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (massDistributionClass), intent(inout)          :: self
  logical                       , intent(in   ), optional :: includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(massDistributionClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (massDistributionClass), intent(inout) :: self
 integer                       , intent(in   ) :: stateFile
 type   (fgsl_file            ), intent(in   ) :: fgslStateFile
 integer(c_size_t             ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (massDistributionClass)                          , intent(inout) :: self
 type     (varying_string       ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                )                           , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`density` Return the density of the distribution at the given coordinates. Must have the following interface:

```
double precision function myImplementationDensity(self,coordinates)
 class(massDistributionClass), intent(inout) :: self
 class(coordinate           ), intent(in   ) :: coordinates
end double precision function myImplementationDensity
```

`symmetry` Return the symmetry of the distribution. Must have the following interface:

```
integer function myImplementationSymmetry(self)
 class(massDistributionClass), intent(inout) :: self
end integer function myImplementationSymmetry
```

`isDimensionless` Return true if the distribution is dimensionless. A default implementation exists. If overridden the following interface must be used:

```
logical function myImplementationIsDimensionless(self)
 class(massDistributionClass), intent(inout) :: self
end logical function myImplementationIsDimensionless
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(massDistributionClass), intent(inout) :: self
 class(massDistributionClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (massDistributionClass), intent(inout) :: self
 integer                       , intent(in   ) :: stateFile
 type   (fgsl_file            ), intent(in   ) :: fgslStateFile
 integer(c_size_t             ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

`densityRadialMoment` Return the radial moment of the distribution. Must have the following interface:

```
 double precision function myImplementationDensityRadialMoment(self,moment,radiusMinimum&
&  , radiusMaximum,isInfinite)
  class           (massDistributionClass), intent(inout)           :: self
  double precision                       , intent(in   )           :: moment
  double precision                       , intent(in   ), optional :: radiusMinimum, &
&  radiusMaximum
  logical                                , intent(  out), optional :: isInfinite
 end double precision function myImplementationDensityRadialMoment
```

Existing implementations are:

`massDistributionHernquist` A Hernquist [1990] mass distribution class.

`massDistributionSersic` A Sérsic mass distribution class.

`massDistributionMiyamotoNagai` An Miyamoto-Nagai model [Miyamoto and Nagai, 1975] mass distribution class.

`massDistributionSpherical` An abstract mass distribution class for spherically symmetric distributions.

`massDistributionExponentialDisk` The exponential disk mass distribution: $\rho(r,z) = \rho_0 \exp(-r/r_\mathrm{s})\mathrm{sech}^2(z/z_\mathrm{s})$.

`massDistributionBetaProfile` An mass distribution class for β-profile distributions.

`massDistributionNFW` An NFW [Navarro et al., 1996] mass distribution class.

`massDistributionCylindrical` An abstract mass distribution class for cylindrically symmetric distributions.

Mass Function Incompletenesses

Additional implementations for mass function incompletenesses are added using the `massFunctionIncompleteness` class. The implementation should be placed in a file containing the directive:

```
!# <massFunctionIncompleteness name="massFunctionIncompletenessMyImplementation">
!# <description>A short description of the implementation.</description>
!# </massFunctionIncompleteness>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `massFunctionIncompletenessMyImplementation`. The file *must* define a type that extends the `massFunctionIncompletenessClass` class (or extends another type which is itself an extension of the `massFunctionIncompletenessClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (massFunctionIncompletenessClass), intent(inout)           :: self
  logical                                 , intent(in   ), optional :: &
& includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (massFunctionIncompletenessClass), intent(inout)           :: self
 type   (inputParameters                ), intent(inout)           :: descriptor
 logical                                 , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(massFunctionIncompletenessClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (massFunctionIncompletenessClass), intent(inout) :: self
 integer                                 , intent(in   ) :: stateFile
 type   (fgsl_file                      ), intent(in   ) :: fgslStateFile
 integer(c_size_t                       ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (massFunctionIncompletenessClass)                          , intent(inout) &
&  :: self
 type     (varying_string                ), allocatable, dimension(:), intent(inout) &
&  :: allowedParameters
 character(len=*                         )                          , intent(in   ) &
&  :: sourceName
end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(massFunctionIncompletenessClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(massFunctionIncompletenessClass), intent(inout) :: self
 class(massFunctionIncompletenessClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (massFunctionIncompletenessClass), intent(inout) :: self
 integer                                 , intent(in   ) :: stateFile
 type   (fgsl_file                       ), intent(in   ) :: fgslStateFile
 integer(c_size_t                        ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

`completeness` Return the completeness of the observational sample at the given mass. Must have the following interface:

```
double precision function myImplementationCompleteness(self,mass)
 class           (massFunctionIncompletenessClass), intent(inout) :: self
 double precision                                 , intent(in   ) :: mass
end double precision function myImplementationCompleteness
```

Existing implementations are:

`massFunctionIncompletenessSurfaceBrightness` Computes incompleteness assuming a normal distribution of surface brightnesses. See §12.31.2.

`massFunctionIncompletenessComplete` Computes incompleteness for a complete survey. See §12.31.1.

Merger Mass Movements

Additional implementations for merger mass movements are added using the `mergerMassMovements` class. The implementation should be placed in a file containing the directive:

```
!# <mergerMassMovements name="mergerMassMovementsMyImplementation">
!# <description>A short description of the implementation.</description>
!# </mergerMassMovements>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `mergerMassMovementsMyImplementation`. The file *must* define a type that extends the `mergerMassMovementsClass` class (or extends another type which is itself an extension of the `mergerMassMovementsClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
 class  (mergerMassMovementsClass), intent(inout)          :: self
 logical                          , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (mergerMassMovementsClass), intent(inout)          :: self
 type   (inputParameters         ), intent(inout)          :: descriptor
 logical                          , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(mergerMassMovementsClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (mergerMassMovementsClass), intent(inout) :: self
 integer                          , intent(in   ) :: stateFile
 type   (fgsl_file                ), intent(in   ) :: fgslStateFile
 integer(c_size_t                 ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (mergerMassMovementsClass)                          , intent(inout) :: self
 type     (varying_string         ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                  )                          , intent(in   ) :: &
&  sourceName
end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(mergerMassMovementsClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`get` Determine movements of mass during mergers. Must have the following interface:

```
subroutine myImplementationGet(self,node,destinationGasSatellite, &
&  destinationStarsSatellite, destinationGasHost, destinationStarsHost,mergerIsMajor)
 class  (mergerMassMovementsClass), intent(inout) :: self
 type   (treeNode                ), intent(inout) :: node
 integer                          , intent(  out) :: destinationGasSatellite, &
&  destinationStarsSatellite, destinationGasHost, destinationStarsHost
 logical                          , intent(  out) :: mergerIsMajor
end subroutine myImplementationGet
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(mergerMassMovementsClass), intent(inout) :: self
 class(mergerMassMovementsClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (mergerMassMovementsClass), intent(inout) :: self
 integer                          , intent(in   ) :: stateFile
 type   (fgsl_file                ), intent(in   ) :: fgslStateFile
 integer(c_size_t                 ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`mergerMassMovementsBaugh2005` A merger mass movements class which uses a simple calculation.

`mergerMassMovementsVerySimple` A merger mass movements class which uses a simple calculation.

`mergerMassMovementsSimple` A merger mass movements class which uses a simple calculation.

Merger Progenitor Properties

Additional implementations for merger progenitor properties are added using the `mergerProgenitorProperties` class. The implementation should be placed in a file containing the directive:

```
!# <mergerProgenitorProperties name="mergerProgenitorPropertiesMyImplementation">
!# <description>A short description of the implementation.</description>
!# </mergerProgenitorProperties>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `mergerProgenitorPropertiesMyImplementation`. The file *must* define a type that extends the `mergerProgenitorPropertiesClass` class (or extends another type which is itself an extension of the `mergerProgenitorPropertiesClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (mergerProgenitorPropertiesClass), intent(inout)          :: self
  logical                                 , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (mergerProgenitorPropertiesClass), intent(inout)          :: self
 type   (inputParameters                ), intent(inout)          :: descriptor
 logical                                 , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(mergerProgenitorPropertiesClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (mergerProgenitorPropertiesClass), intent(inout) :: self
 integer                                 , intent(in   ) :: stateFile
 type   (fgsl_file                       ), intent(in   ) :: fgslStateFile
 integer(c_size_t                        ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (mergerProgenitorPropertiesClass)                          , intent(inout) &
&  :: self
 type     (varying_string                ), allocatable, dimension(:), intent(inout) &
&  :: allowedParameters
 character(len=*                         )                          , intent(in   ) &
&  :: sourceName
end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(mergerProgenitorPropertiesClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`get` alculates progenitor properties for merger calculations. Must have the following interface:

```
subroutine myImplementationGet(self,nodeSatellite            , nodeHost,massSatellite  &
&           , massHost,massSpheroidSatellite    , massSpheroidHost,&
&  massSpheroidHostPreMerger, radiusSatellite,radiusHost               , &
&  factorAngularMomentum,massSpheroidRemnant      , massGasSpheroidRemnant)
 class           (mergerProgenitorPropertiesClass), intent(inout)         :: self
 type            (treeNode                       ), intent(inout), target :: &
&  nodeSatellite, nodeHost
 double precision                                  , intent(  out)         :: &
&  massSatellite, massHost
 double precision                                  , intent(  out)         :: &
&  massSpheroidSatellite, massSpheroidHost
 double precision                                  , intent(  out)         :: &
&  massSpheroidHostPreMerger, radiusSatellite
 double precision                                  , intent(  out)         :: radiusHost&
&  , factorAngularMomentum
 double precision                                  , intent(  out)         :: &
&  massSpheroidRemnant, massGasSpheroidRemnant
end subroutine myImplementationGet
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(mergerProgenitorPropertiesClass), intent(inout) :: self
 class(mergerProgenitorPropertiesClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (mergerProgenitorPropertiesClass), intent(inout) :: self
 integer                                 , intent(in   ) :: stateFile
 type   (fgsl_file                       ), intent(in   ) :: fgslStateFile
```

```
 integer(c_size_t                      ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`mergerProgenitorPropertiesSimple` A merger progenitor properties class which uses a simple calculation.

`mergerProgenitorPropertiesCole2000` A merger progenitor properties class which uses the algorithm of Cole et al. [2000].

`mergerProgenitorPropertiesStandard` A merger progenitor properties class which uses a standard calculation.

Merger Remnant Sizes

Additional implementations for merger remnant sizes are added using the `mergerRemnantSize` class. The implementation should be placed in a file containing the directive:

```
!# <mergerRemnantSize name="mergerRemnantSizeMyImplementation">
!# <description>A short description of the implementation.</description>
!# </mergerRemnantSize>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `mergerRemnantSize-MyImplementation`. The file *must* define a type that extends the `mergerRemnantSizeClass` class (or extends another type which is itself an extension of the `mergerRemnantSizeClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (mergerRemnantSizeClass), intent(inout)           :: self
  logical                        , intent(in   ), optional :: includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (mergerRemnantSizeClass), intent(inout)           :: self
 type   (inputParameters       ), intent(inout)           :: descriptor
 logical                        , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(mergerRemnantSizeClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (mergerRemnantSizeClass), intent(inout) :: self
 integer                        , intent(in   ) :: stateFile
 type   (fgsl_file             ), intent(in   ) :: fgslStateFile
 integer(c_size_t              ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (mergerRemnantSizeClass)                              , intent(inout) :: self
 type     (varying_string        ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                 )                              , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(mergerRemnantSizeClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`get` Determine merger remnant size and related properties. Must have the following interface:

```
subroutine myImplementationGet(self,node,radius, velocityCircular, &
&  angularMomentumSpecific)
 class           (mergerRemnantSizeClass), intent(inout) :: self
 type            (treeNode              ), intent(inout) :: node
 double precision                         , intent(  out) :: radius, velocityCircular, &
&  angularMomentumSpecific
 end subroutine myImplementationGet
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(mergerRemnantSizeClass), intent(inout) :: self
 class(mergerRemnantSizeClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (mergerRemnantSizeClass), intent(inout) :: self
 integer                        , intent(in   ) :: stateFile
 type   (fgsl_file             ), intent(in   ) :: fgslStateFile
 integer(c_size_t              ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`mergerRemnantSizeCovington2008` A merger remnant size class which uses the Cole et al. [2000] algorithm.

`mergerRemnantSizeNull` A merger remnant size class which takes no action.

`mergerRemnantSizeCole2000` A merger remnant size class which uses the Cole et al. [2000] algorithm.

Merger Tree Branching Probabilities

Additional implementations for merger tree branching probabilities are added using the `mergerTreeBranchingProbability` class. The implementation should be placed in a file containing the directive:

```
!# <mergerTreeBranchingProbability name="mergerTreeBranchingProbabilityMyImplementation">
!# <description>A short description of the implementation.</description>
!# </mergerTreeBranchingProbability>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `mergerTreeBranchingProbabilityMyImplementation`. The file *must* define a type that extends the `mergerTreeBranchingProbabilityClass` class (or extends another type which is itself an extension of the `mergerTreeBranchingProbabilityClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`massBranch` Returns the mass of a new halo created by a branching event. Must have the following interface:

```
 double precision function myImplementationMassBranch(self,haloMass             , &
&  deltaCritical, massResolution, probabilityFraction,randomNumberGenerator,node)
  class           (mergerTreeBranchingProbabilityClass), intent(inout)         :: self
  double precision                                     , intent(in   )         :: &
&  haloMass, deltaCritical, massResolution, probabilityFraction
  type            (pseudoRandom                       ), intent(inout)         :: &
&  randomNumberGenerator
  type            (treeNode                           ), intent(inout), target :: node
 end double precision function myImplementationMassBranch
```

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (mergerTreeBranchingProbabilityClass), intent(inout)          :: self
  logical                                     , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (mergerTreeBranchingProbabilityClass), intent(inout)          :: self
 type   (inputParameters                    ), intent(inout)          :: descriptor
 logical                                     , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`stepMaximum` Returns the maximum step in "time" allowed by this algorithm. Must have the following interface:

```
 double precision function myImplementationStepMaximum(self,haloMass, deltaCritical, &
&  massResolution)
  class           (mergerTreeBranchingProbabilityClass), intent(inout) :: self
  double precision                                     , intent(in   ) :: haloMass, &
&  deltaCritical, massResolution
 end double precision function myImplementationStepMaximum
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(mergerTreeBranchingProbabilityClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (mergerTreeBranchingProbabilityClass), intent(inout) :: self
 integer                                     , intent(in   ) :: stateFile
 type   (fgsl_file                          ), intent(in   ) :: fgslStateFile
 integer(c_size_t                           ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`probability` Computes the probability per unit "time" that a branching event occurs. Must have the following interface:

```
 double precision function myImplementationProbability(self,haloMass, deltaCritical, &
&  massResolution,node)
  class           (mergerTreeBranchingProbabilityClass), intent(inout)         :: self
  double precision                                     , intent(in   )         :: &
&  haloMass, deltaCritical, massResolution
  type            (treeNode                          ), intent(inout), target :: node
 end double precision function myImplementationProbability
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (mergerTreeBranchingProbabilityClass)                          , intent(&
&  inout) :: self
 type     (varying_string                    ), allocatable, dimension(:), intent(&
&  inout) :: allowedParameters
 character(len=*                             )                          , intent(in  &
&   ) :: sourceName
end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(mergerTreeBranchingProbabilityClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(mergerTreeBranchingProbabilityClass), intent(inout) :: self
 class(mergerTreeBranchingProbabilityClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`probabilityBound` Computes a bound (upper or lower) to the probability per unit "time" that a branching event occurs. Must have the following interface:

```
double precision function myImplementationProbabilityBound(self,haloMass, deltaCritical&
&  , massResolution,bound,node)
 class           (mergerTreeBranchingProbabilityClass), intent(inout)         :: self
 double precision                                     , intent(in   )         :: &
&  haloMass, deltaCritical, massResolution
 integer                                              , intent(in   )         :: bound
 type            (treeNode                           ), intent(inout), target :: node
end double precision function myImplementationProbabilityBound
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (mergerTreeBranchingProbabilityClass), intent(inout) :: self
 integer                                     , intent(in   ) :: stateFile
 type   (fgsl_file                           ), intent(in   ) :: fgslStateFile
 integer(c_size_t                            ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

`fractionSubresolution` Computes the fraction of subresolution mass accreted per unit "time" Must have the following interface:

```
double precision function myImplementationFractionSubresolution(self,haloMass, &
&  deltaCritical, massResolution,node)
 class           (mergerTreeBranchingProbabilityClass), intent(inout)         :: self
```

```
  double precision                                        , intent(in   )         :: &
&  haloMass, deltaCritical, massResolution
  type            (treeNode                             ), intent(inout), target :: node
 end double precision function myImplementationFractionSubresolution
```

Existing implementations are:

`mergerTreeBranchingProbabilityParkinsonColeHelly` Merger tree branching probabilities using the algorithm of Parkinson et al. [2008]..

`mergerTreeBranchingProbabilityGnrlzdPrssSchchtr` Merger tree branching probabilities using a generalized Press-Schechter approach.

Modifiers for merger tree branching probabilities

Additional implementations for modifiers for merger tree branching probabilities are added using the `mergerTreeBranchingProbabilityModifier` class. The implementation should be placed in a file containing the directive:

```
!# <mergerTreeBranchingProbabilityModifier name="mergerTreeBranchingProbabilityModifierMyImplementation"
!# <description>A short description of the implementation.</description>
!# </mergerTreeBranchingProbabilityModifier>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `mergerTreeBranchingProbabilityModifierMyImplementation`. The file *must* define a type that extends the `mergerTreeBranchingProbabilityModifierClass` class (or extends another type which is itself an extension of the `mergerTreeBranchingProbabilityModifierClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (mergerTreeBranchingProbabilityModifierClass), intent(inout)           :: self
  logical                                             , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationDescriptor(self,descriptor,includeMethod)
  class  (mergerTreeBranchingProbabilityModifierClass), intent(inout)           :: self
  type   (inputParameters                             ), intent(inout)           :: &
&  descriptor
  logical                                             , intent(in   ), optional :: &
&  includeMethod
 end subroutine myImplementationDescriptor
```

`rateModifier` Return the multiplicative modifier to the tree branch probability rate. Must have the following interface:

```
 double precision function myImplementationRateModifier(self,deltaParent, sigmaChild, &
&  sigmaParent)
  class           (mergerTreeBranchingProbabilityModifierClass), intent(inout) :: self
  double precision                                             , intent(in   ) :: &
&  deltaParent, sigmaChild, sigmaParent
 end double precision function myImplementationRateModifier
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAutoHook(self)
  class(mergerTreeBranchingProbabilityModifierClass), intent(inout) :: self
 end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
  class  (mergerTreeBranchingProbabilityModifierClass), intent(inout) :: self
  integer                                             , intent(in   ) :: stateFile
  type   (fgsl_file                                  ), intent(in   ) :: fgslStateFile
  integer(c_size_t                                   ), intent(in   ) :: &
&  stateOperationID
 end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
  class    (mergerTreeBranchingProbabilityModifierClass)                          , &
&  intent(inout) :: self
  type     (varying_string                             ), allocatable, dimension(:), &
&  intent(inout) :: allowedParameters
  character(len=*                                      )                          , &
&  intent(in   ) :: sourceName
 end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationObjectType(self)
  class(mergerTreeBranchingProbabilityModifierClass), intent(inout) :: self
 end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationDeepCopy(self,destination)
  class(mergerTreeBranchingProbabilityModifierClass), intent(inout) :: self
  class(mergerTreeBranchingProbabilityModifierClass), intent(inout) :: destination
 end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (mergerTreeBranchingProbabilityModifierClass), intent(inout) :: self
 integer                                             , intent(in   ) :: stateFile
 type   (fgsl_file                                   ), intent(in   ) :: fgslStateFile
 integer(c_size_t                                    ), intent(in   ) :: &
&  stateOperationID
 end subroutine myImplementationStateRestore
```

Existing implementations are:

`mergerTreeBranchingProbabilityModifierIdentity` Provides a merger tree branching probability rate modifier which always returns the identity modifier.

`mergerTreeBranchingProbabilityModifierParkinson2008` Provides a merger tree branching probability rate modifier which uses the model of Parkinson et al. [2008].

Merger Tree Mass Distributions

Additional implementations for merger tree mass distributions are added using the `mergerTreeBuildMassDistribution` class. The implementation should be placed in a file containing the directive:

```
!# <mergerTreeBuildMassDistribution name="mergerTreeBuildMassDistributionMyImplementation">
!# <description>A short description of the implementation.</description>
!# </mergerTreeBuildMassDistribution>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `mergerTreeBuildMassDistributionMyImplementation`. The file *must* define a type that extends the `mergerTreeBuildMassDistributionClass` class (or extends another type which is itself an extension of the `mergerTreeBuildMassDistributionClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
 class  (mergerTreeBuildMassDistributionClass), intent(inout)           :: self
 logical                                      , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (mergerTreeBuildMassDistributionClass), intent(inout)           :: self
```

```
     type   (inputParameters                    ), intent(inout)          :: descriptor
     logical                                     , intent(in   ), optional :: &
    &  includeMethod
     end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(mergerTreeBuildMassDistributionClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (mergerTreeBuildMassDistributionClass), intent(inout) :: self
 integer                                      , intent(in   ) :: stateFile
 type   (fgsl_file                            ), intent(in   ) :: fgslStateFile
 integer(c_size_t                             ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (mergerTreeBuildMassDistributionClass)                          , intent(&
&  inout) :: self
 type     (varying_string                      ), allocatable, dimension(:), intent(&
&  inout) :: allowedParameters
 character(len=*                               )                          , intent(in &
&    ) :: sourceName
 end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(mergerTreeBuildMassDistributionClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(mergerTreeBuildMassDistributionClass), intent(inout) :: self
 class(mergerTreeBuildMassDistributionClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`sample` Returns the sampling rate for merger trees of the given `mass`, per decade of halo mass. Must have the following interface:

```
 double precision function myImplementationSample(self,mass,time,massMinimum,massMaximum&
&  )
  class           (mergerTreeBuildMassDistributionClass), intent(inout) :: self
  double precision                                      , intent(in   ) :: mass, time, &
&  massMinimum, massMaximum
 end double precision function myImplementationSample
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (mergerTreeBuildMassDistributionClass), intent(inout) :: self
 integer                                      , intent(in   ) :: stateFile
 type   (fgsl_file                           ), intent(in   ) :: fgslStateFile
 integer(c_size_t                            ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`mergerTreeBuildMassDistributionStllrMssFnctn` A merger tree halo mass function sampling class optimized to minimize variance in the model stellar mass function.

`mergerTreeBuildMassDistributionPowerLaw` A merger tree halo mass function sampling class in which the sampling rate is given by a power-law in halo mass.

`mergerTreeBuildMassDistributionHaloMassFunction` A merger tree halo mass function sampling class in which the sampling rate is proportional to the halo mass function.

`mergerTreeBuildMassDistributionGaussian` A merger tree halo mass function sampling class in which the sampling rate is given by a Gaussian distribution in halo mass.

Merger Tree Build Masses

Additional implementations for merger tree build masses are added using the `mergerTreeBuildMasses` class. The implementation should be placed in a file containing the directive:

```
!# <mergerTreeBuildMasses name="mergerTreeBuildMassesMyImplementation">
!# <description>A short description of the implementation.</description>
!# </mergerTreeBuildMasses>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `mergerTreeBuildMassesMyImplementation`. The file *must* define a type that extends the `mergerTreeBuildMassesClass` class (or extends another type which is itself an extension of the `mergerTreeBuildMassesClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (mergerTreeBuildMassesClass), intent(inout)          :: self
  logical                            , intent(in   ), optional :: includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (mergerTreeBuildMassesClass), intent(inout)          :: self
 type   (inputParameters           ), intent(inout)          :: descriptor
 logical                            , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(mergerTreeBuildMassesClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (mergerTreeBuildMassesClass), intent(inout) :: self
 integer                            , intent(in   ) :: stateFile
 type   (fgsl_file                  ), intent(in   ) :: fgslStateFile
 integer(c_size_t                   ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (mergerTreeBuildMassesClass)                          , intent(inout) :: &
&  self
 type     (varying_string            ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                     )                          , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`construct` Returns a set of merger tree masses (and either their weights, or corresponding mass interval) to be built. Must have the following interface:

```
subroutine myImplementationConstruct(self,time,mass, massMinimum, massMaximum, weight)
 class           (mergerTreeBuildMassesClass)                          , intent(inout)&
&   :: self
 double precision                                                      , intent(in   )&
&   :: time
 double precision                           , allocatable, dimension(:), intent(  out)&
&   :: mass, massMinimum, massMaximum, weight
 end subroutine myImplementationConstruct
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(mergerTreeBuildMassesClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(mergerTreeBuildMassesClass), intent(inout) :: self
 class(mergerTreeBuildMassesClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (mergerTreeBuildMassesClass), intent(inout) :: self
 integer                            , intent(in   ) :: stateFile
 type   (fgsl_file                 ), intent(in   ) :: fgslStateFile
 integer(c_size_t                  ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`mergerTreeBuildMassesSampledDistribution` A merger tree masses class which samples masses from a distribution.

`mergerTreeBuildMassesReadHDF5` A merger tree masses class which reads masses from an HDF5 file.

`mergerTreeBuildMassesFixedMass` A merger tree masses class which uses a fixed mass for trees.

`mergerTreeBuildMassesSampledDistributionQuasiRandom` A merger tree masses class which samples masses from a distribution using quasi-random sampling.

`mergerTreeBuildMassesReadXML` A merger tree masses class which reads masses from an XML file.

`mergerTreeBuildMassesSampledDistributionUniform` A merger tree masses class which samples masses from a distribution uniformly.

`mergerTreeBuildMassesUnion` A merger tree masses class which constructs the union of other classes.

`mergerTreeBuildMassesRead` A merger tree masses class which samples masses from a distribution.

`mergerTreeBuildMassesSampledDistributionPseudoRandom` A merger tree masses class which samples masses from a distribution using pseudo-random sampling.

Merger Tree Builders

Additional implementations for merger tree builders are added using the `mergerTreeBuilder` class. The implementation should be placed in a file containing the directive:

```
!# <mergerTreeBuilder name="mergerTreeBuilderMyImplementation">
!# <description>A short description of the implementation.</description>
!# </mergerTreeBuilder>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `mergerTreeBuilderMyImplementation`. The file *must* define a type that extends the `mergerTreeBuilderClass` class (or extends another type which is itself an extension of the `mergerTreeBuilderClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
 class  (mergerTreeBuilderClass), intent(inout)          :: self
 logical                        , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (mergerTreeBuilderClass), intent(inout)          :: self
 type   (inputParameters       ), intent(inout)          :: descriptor
 logical                        , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(mergerTreeBuilderClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (mergerTreeBuilderClass), intent(inout) :: self
 integer                        , intent(in   ) :: stateFile
 type   (fgsl_file              ), intent(in   ) :: fgslStateFile
 integer(c_size_t               ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (mergerTreeBuilderClass)                          , intent(inout) :: self
 type     (varying_string        ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                 )                          , intent(in   ) :: &
&  sourceName
end subroutine myImplementationAllowedParameters
```

`build` Builds and returns a merger tree given the root `node`. Must have the following interface:

```
subroutine myImplementationBuild(self,tree)
 class(mergerTreeBuilderClass), intent(inout)         :: self
 type (mergerTree            ), intent(inout), target :: tree
end subroutine myImplementationBuild
```

`timeEarliestSet` Set the earliest time for the builder to the given value. Must have the following interface:

```
subroutine myImplementationTimeEarliestSet(self,timeEarliest)
 class           (mergerTreeBuilderClass), intent(inout) :: self
 double precision                        , intent(in   ) :: timeEarliest
end subroutine myImplementationTimeEarliestSet
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(mergerTreeBuilderClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(mergerTreeBuilderClass), intent(inout) :: self
 class(mergerTreeBuilderClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (mergerTreeBuilderClass), intent(inout) :: self
 integer                        , intent(in   ) :: stateFile
 type   (fgsl_file              ), intent(in   ) :: fgslStateFile
 integer(c_size_t               ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`mergerTreeBuilderCole2000` Merger trees are built using the algorithm of Cole et al. [2000].

Merger Tree Constructors

Additional implementations for merger tree constructors are added using the `mergerTreeConstructor` class. The implementation should be placed in a file containing the directive:

```
!# <mergerTreeConstructor name="mergerTreeConstructorMyImplementation">
!# <description>A short description of the implementation.</description>
!# </mergerTreeConstructor>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `mergerTreeConstructorMyImplementation`. The file *must* define a type that extends the `mergerTreeConstructorClass` class (or extends another type which is itself an extension of the `mergerTreeConstructorClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
 class  (mergerTreeConstructorClass), intent(inout)           :: self
 logical                            , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (mergerTreeConstructorClass), intent(inout)           :: self
 type   (inputParameters           ), intent(inout)           :: descriptor
 logical                            , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(mergerTreeConstructorClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (mergerTreeConstructorClass), intent(inout) :: self
 integer                            , intent(in   ) :: stateFile
 type   (fgsl_file                  ), intent(in   ) :: fgslStateFile
 integer(c_size_t                   ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (mergerTreeConstructorClass)                          , intent(inout) :: &
&  self
 type     (varying_string            ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                     )                          , intent(in   ) :: &
&  sourceName
end subroutine myImplementationAllowedParameters
```

`construct` Construct the merger tree corresponding to the given `treeNumber`. Must have the following interface:

```
type(mergerTree), pointer function myImplementationConstruct(self,treeNumber)
 class  (mergerTreeConstructorClass), intent(inout) :: self
 integer(c_size_t                  ), intent(in   ) :: treeNumber
end type(mergerTree), pointer function myImplementationConstruct
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(mergerTreeConstructorClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(mergerTreeConstructorClass), intent(inout) :: self
 class(mergerTreeConstructorClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (mergerTreeConstructorClass), intent(inout) :: self
 integer                            , intent(in   ) :: stateFile
 type   (fgsl_file                 ), intent(in   ) :: fgslStateFile
 integer(c_size_t                  ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`mergerTreeConstructorFullySpecified` Merger tree constructor class which constructs a merger tree given a full specification in XML.

`mergerTreeConstructorRead` Merger tree constructor class which builds merger trees assuming smooth accretion. See §12.32.1.

`mergerTreeConstructorStateRestored` Merger tree constructor class which constructs a merger tree by restoring state from file.

`mergerTreeConstructorSmoothAccretion` Merger tree constructor class which builds merger trees assuming smooth accretion.

`mergerTreeConstructorBuild` Merger tree constructor class which builds merger trees.

Merger Tree Evolution Timesteps

Additional implementations for merger tree evolution timesteps are added using the `mergerTreeEvolveTimestep` class. The implementation should be placed in a file containing the directive:

```
!# <mergerTreeEvolveTimestep name="mergerTreeEvolveTimestepMyImplementation">
!# <description>A short description of the implementation.</description>
!# </mergerTreeEvolveTimestep>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `mergerTreeEvolveTimestepMyImplementation`. The file *must* define a type that extends the `mergerTreeEvolveTimestepClass` class (or extends another type which is itself an extension of the `mergerTreeEvolveTimestepClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
 class  (mergerTreeEvolveTimestepClass), intent(inout)           :: self
 logical                               , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (mergerTreeEvolveTimestepClass), intent(inout)           :: self
 type   (inputParameters              ), intent(inout)           :: descriptor
 logical                               , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(mergerTreeEvolveTimestepClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (mergerTreeEvolveTimestepClass), intent(inout) :: self
 integer                               , intent(in   ) :: stateFile
 type   (fgsl_file                    ), intent(in   ) :: fgslStateFile
 integer(c_size_t                     ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
  class    (mergerTreeEvolveTimestepClass)                          , intent(inout) :: &
&  self
  type     (varying_string              ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
  character(len=*                       )                          , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(mergerTreeEvolveTimestepClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(mergerTreeEvolveTimestepClass), intent(inout) :: self
 class(mergerTreeEvolveTimestepClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (mergerTreeEvolveTimestepClass), intent(inout) :: self
 integer                               , intent(in   ) :: stateFile
 type   (fgsl_file                     ), intent(in   ) :: fgslStateFile
 integer(c_size_t                      ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

`timeEvolveTo` Return the time to which the node can be evolved. Must have the following interface:

```
 double precision function myImplementationTimeEvolveTo(self,node,task,taskSelf,report,&
&  lockNode,lockType)
  class    (mergerTreeEvolveTimestepClass), intent(inout)          :: self
  type     (treeNode                     ), intent(inout), target  :: node
  procedure(timestepTask                 ), intent(  out), pointer :: task
  class    (*                            ), intent(  out), pointer :: taskSelf
  logical                                  , intent(in   )          :: report
  type     (treeNode                     ), intent(  out), pointer :: lockNode
  type     (varying_string               ), intent(  out)          :: lockType
 end double precision function myImplementationTimeEvolveTo
```

Existing implementations are:

`mergerTreeEvolveTimestepMulti` A merger tree evolution timestepping class which takes the minimum over multiple other timesteppers.

`mergerTreeEvolveTimestepRecordEvolution` A merger tree evolution timestepping class which limits the step to the next epoch at which to record evolution of the main branch galaxy. See §8.1.3.

`mergerTreeEvolveTimestepHistory` A merger tree evolution timestepping class which limits the step the next epoch at which to store global history. See §8.1.3.

`mergerTreeEvolveTimestepSimple` A merger tree evolution timestepping class which limits the step to a fraction of the current time or an absolute step, whichever is smaller. See §8.1.2.

`mergerTreeEvolveTimestepSatellite` A merger tree evolution timestepping class which limits the step to the next satellite merger. See §8.1.2.

`mergerTreeEvolveTimestepStandard` A merger tree evolution timestepping class which limits the step to the minimum of that given by the `simple` and `satellite` timesteps.

Merger Tree Evolvers

Additional implementations for merger tree evolvers are added using the `mergerTreeEvolver` class. The implementation should be placed in a file containing the directive:

```
!# <mergerTreeEvolver name="mergerTreeEvolverMyImplementation">
!# <description>A short description of the implementation.</description>
!# </mergerTreeEvolver>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `mergerTreeEvolverMyImplementation`. The file *must* define a type that extends the `mergerTreeEvolverClass` class (or extends another type which is itself an extension of the `mergerTreeEvolverClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (mergerTreeEvolverClass), intent(inout)           :: self
  logical                        , intent(in   ), optional :: includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationDescriptor(self,descriptor,includeMethod)
  class  (mergerTreeEvolverClass), intent(inout)           :: self
  type   (inputParameters       ), intent(inout)           :: descriptor
  logical                        , intent(in   ), optional :: includeMethod
 end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAutoHook(self)
  class(mergerTreeEvolverClass), intent(inout) :: self
 end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (mergerTreeEvolverClass), intent(inout) :: self
 integer                        , intent(in   ) :: stateFile
 type   (fgsl_file             ), intent(in   ) :: fgslStateFile
 integer(c_size_t              ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`evolve` Evolve a merger tree. Must have the following interface:

```
 subroutine myImplementationEvolve(self,tree,timeEnd,treeDidEvolve     , suspendTree,&
&  deadlockReporting,systemClockMaximum,initializationLock,status)
  class           (mergerTreeEvolverClass), intent(inout)           :: self
  type            (mergerTree            ), intent(inout), target   :: tree
  double precision                         , intent(in   )           :: timeEnd
  logical                                  , intent(  out)           :: treeDidEvolve, &
&  suspendTree
  logical                                  , intent(in   )           :: deadlockReporting
  integer         (kind_int8             ), intent(in   ), optional :: &
&  systemClockMaximum
  integer         (omp_lock_kind         ), intent(inout), optional :: &
&  initializationLock
  integer                                  , intent(  out), optional :: status
 end subroutine myImplementationEvolve
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
  class    (mergerTreeEvolverClass)                          , intent(inout) :: self
  type     (varying_string        ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
  character(len=*                 )                          , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(mergerTreeEvolverClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(mergerTreeEvolverClass), intent(inout) :: self
 class(mergerTreeEvolverClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (mergerTreeEvolverClass), intent(inout) :: self
 integer                        , intent(in   ) :: stateFile
 type   (fgsl_file             ), intent(in   ) :: fgslStateFile
 integer(c_size_t              ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`mergerTreeEvolverNonEvolving` A non-evolving merger tree evolver.

`mergerTreeEvolverStandard` The standard merger tree evolver.

Merger Tree Importer

Additional implementations for merger tree importer are added using the `mergerTreeImporter` class. The implementation should be placed in a file containing the directive:

```
!# <mergerTreeImporter name="mergerTreeImporterMyImplementation">
!# <description>A short description of the implementation.</description>
!# </mergerTreeImporter>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `mergerTreeImporterMyImplementation`. The file *must* define a type that extends the `mergerTreeImporterClass` class (or extends another type which is itself an extension of the `mergerTreeImporterClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`positionsAvailable` Return true if positions and/or velocities are available. Must have the following interface:

```
logical function myImplementationPositionsAvailable(self,positions, velocities)
 class  (mergerTreeImporterClass), intent(inout) :: self
 logical                         , intent(in   ) :: positions, velocities
end logical function myImplementationPositionsAvailable
```

`treesAreSelfContained` Returns a Boolean integer specifying whether trees are self-contained. Must have the following interface:

```
integer function myImplementationTreesAreSelfContained(self)
 class(mergerTreeImporterClass), intent(inout) :: self
end integer function myImplementationTreesAreSelfContained
```

`cubeLength` Returns the length of the simulation cube. Must have the following interface:

```
double precision function myImplementationCubeLength(self,time,status)
 class           (mergerTreeImporterClass), intent(inout)           :: self
 double precision                         , intent(in   )           :: time
 integer                                  , intent(  out), optional :: status
end double precision function myImplementationCubeLength
```

`nodeCount` Returns the number of nodes in the $i^{\rm th}$ tree. Must have the following interface:

```
integer(kind=c_size_t) function myImplementationNodeCount(self,i)
 class  (mergerTreeImporterClass), intent(inout) :: self
 integer                         , intent(in   ) :: i
end integer(kind=c_size_t) function myImplementationNodeCount
```

`angularMomentaIncludeSubhalos` Returns a Boolean specifying whether halo angular momenta (or spins) include the contribution from their subhalos. Must have the following interface:

```
logical function myImplementationAngularMomentaIncludeSubhalos(self)
 class(mergerTreeImporterClass), intent(inout) :: self
end logical function myImplementationAngularMomentaIncludeSubhalos
```

`positionsArePeriodic` Returns a Boolean integer specifying whether positions are periodic. Must have the following interface:

```
integer function myImplementationPositionsArePeriodic(self)
 class(mergerTreeImporterClass), intent(inout) :: self
end integer function myImplementationPositionsArePeriodic
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (mergerTreeImporterClass), intent(inout)           :: self
 type   (inputParameters        ), intent(inout)           :: descriptor
 logical                         , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (mergerTreeImporterClass), intent(inout) :: self
 integer                         , intent(in   ) :: stateFile
 type   (fgsl_file               ), intent(in   ) :: fgslStateFile
 integer(c_size_t                ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`particleCountAvailable` Return true if particle counts are available. Must have the following interface:

```
logical function myImplementationParticleCountAvailable(self)
 class(mergerTreeImporterClass), intent(inout) :: self
end logical function myImplementationParticleCountAvailable
```

`treeCount` Returns a count of the number of trees available. Must have the following interface:

```
integer(kind=c_size_t) function myImplementationTreeCount(self)
 class(mergerTreeImporterClass), intent(inout) :: self
end integer(kind=c_size_t) function myImplementationTreeCount
```

`massesIncludeSubhalos` Returns a Boolean specifying whether halo masses include the contribution from their subhalos. Must have the following interface:

```
logical function myImplementationMassesIncludeSubhalos(self)
 class(mergerTreeImporterClass), intent(inout) :: self
end logical function myImplementationMassesIncludeSubhalos
```

`open` Opens the file. Must have the following interface:

```
subroutine myImplementationOpen(self,fileName)
 class(mergerTreeImporterClass), intent(inout) :: self
 type (varying_string         ), intent(in   ) :: fileName
end subroutine myImplementationOpen
```

`angularMomentaAvailable` Return true if angular momenta (magnitudes) are available. Must have the following interface:

```
logical function myImplementationAngularMomentaAvailable(self)
 class(mergerTreeImporterClass), intent(inout) :: self
end logical function myImplementationAngularMomentaAvailable
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(mergerTreeImporterClass), intent(inout) :: self
 class(mergerTreeImporterClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (mergerTreeImporterClass), intent(inout) :: self
 integer                         , intent(in   ) :: stateFile
 type   (fgsl_file              ), intent(in   ) :: fgslStateFile
 integer(c_size_t               ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

`treesHaveSubhalos` Returns a Boolean integer specifying whether or not the trees have subhalos. Must have the following interface:

```
integer function myImplementationTreesHaveSubhalos(self)
 class(mergerTreeImporterClass), intent(inout) :: self
end integer function myImplementationTreesHaveSubhalos
```

`angularMomenta3DAvailable` Return true if angular momenta (vectors) are available. Must have the following interface:

```
logical function myImplementationAngularMomenta3DAvailable(self)
 class(mergerTreeImporterClass), intent(inout) :: self
end logical function myImplementationAngularMomenta3DAvailable
```

`subhaloTraceCount` Returns the length of a node's subhalo trace. Must have the following interface:

```
integer(kind=c_size_t) function myImplementationSubhaloTraceCount(self,node)
 class(mergerTreeImporterClass), intent(inout) :: self
 class(nodeData               ), intent(in   ) :: node
end integer(kind=c_size_t) function myImplementationSubhaloTraceCount
```

`velocitiesIncludeHubbleFlow` Returns a Boolean integer specifying whether velocities include the Hubble flow. Must have the following interface:

```
integer function myImplementationVelocitiesIncludeHubbleFlow(self)
 class(mergerTreeImporterClass), intent(inout) :: self
end integer function myImplementationVelocitiesIncludeHubbleFlow
```

`close` Closes the file. Must have the following interface:

```
subroutine myImplementationClose(self)
 class(mergerTreeImporterClass), intent(inout) :: self
end subroutine myImplementationClose
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(mergerTreeImporterClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`canReadSubsets` Returns true if arbitrary subsets of halos from a forest can be read. Must have the following interface:

```
logical function myImplementationCanReadSubsets(self)
 class(mergerTreeImporterClass), intent(inout) :: self
end logical function myImplementationCanReadSubsets
```

`treeWeight` Returns the weight to assign to the $i^{\rm th}$ tree. Must have the following interface:

```
double precision function myImplementationTreeWeight(self,i)
 class  (mergerTreeImporterClass), intent(inout) :: self
 integer                         , intent(in   ) :: i
end double precision function myImplementationTreeWeight
```

`velocityDispersionAvailable` Return true if halo velocity dispersions are available. Must have the following interface:

```
logical function myImplementationVelocityDispersionAvailable(self)
 class(mergerTreeImporterClass), intent(inout) :: self
end logical function myImplementationVelocityDispersionAvailable
```

`scaleRadiiAvailable` Return true if scale radii are available. Must have the following interface:

```
logical function myImplementationScaleRadiiAvailable(self)
 class(mergerTreeImporterClass), intent(inout) :: self
end logical function myImplementationScaleRadiiAvailable
```

`treeIndex` Returns the index of the $i^{\rm th}$ tree. Must have the following interface:

```
integer(kind=kind_int8) function myImplementationTreeIndex(self,i)
 class  (mergerTreeImporterClass), intent(inout) :: self
 integer                         , intent(in   ) :: i
end integer(kind=kind_int8) function myImplementationTreeIndex
```

`spinAvailable` Return true if spin (magnitudes) are available. Must have the following interface:

```
logical function myImplementationSpinAvailable(self)
 class(mergerTreeImporterClass), intent(inout) :: self
end logical function myImplementationSpinAvailable
```

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
 class  (mergerTreeImporterClass), intent(inout)           :: self
 logical                         , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`spin3DAvailable` Return true if spin (vectors) are available. Must have the following interface:

```
logical function myImplementationSpin3DAvailable(self)
 class(mergerTreeImporterClass), intent(inout) :: self
end logical function myImplementationSpin3DAvailable
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(mergerTreeImporterClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`subhaloTrace` Supplies epochs, positions, and velocities for traced subhalos. Must have the following interface:

```
subroutine myImplementationSubhaloTrace(self,node,time,position, velocity)
 class           (mergerTreeImporterClass)               , intent(inout) :: self
 class           (nodeData               )               , intent(in   ) :: node
 double precision                         , dimension(:)  , intent(  out) :: time
 double precision                         , dimension(:,:), intent(  out) :: position, &
&  velocity
end subroutine myImplementationSubhaloTrace
```

`velocityMaximumAvailable` Return true if rotation curve velocity maxima are available. Must have the following interface:

```
logical function myImplementationVelocityMaximumAvailable(self)
 class(mergerTreeImporterClass), intent(inout) :: self
end logical function myImplementationVelocityMaximumAvailable
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (mergerTreeImporterClass)                          , intent(inout) :: self
 type     (varying_string         ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                  )                          , intent(in   ) :: &
&  sourceName
end subroutine myImplementationAllowedParameters
```

`import` Imports the $i^{\rm th}$ tree. Must have the following interface:

```
 subroutine myImplementationImport(self,i,nodes,nodeSubset,requireScaleRadii, &
&  requireAngularMomenta, requireAngularMomenta3D, requireSpin, requireSpin3D, &
&  requirePositions, structureOnly,requireNamedReals, requireNamedIntegers)
  class  (mergerTreeImporterClass)                              , intent(inout)           &
&  :: self
  integer                                                       , intent(in   )           &
&  :: i
  class  (nodeDataMinimal       ), allocatable, dimension(:), intent(  out)           &
&  :: nodes
  integer(c_size_t              )             , dimension(:), intent(in   ), optional &
&  :: nodeSubset
  logical                                                       , intent(in   ), optional &
&  :: requireScaleRadii, requireAngularMomenta, requireAngularMomenta3D, requireSpin, &
&  requireSpin3D, requirePositions, structureOnly
  type   (varying_string        )             , dimension(:), intent(in   ), optional &
&  :: requireNamedReals, requireNamedIntegers
 end subroutine myImplementationImport
```

Existing implementations are:

`mergerTreeImporterSussing` Importer for "Sussing Merger Trees" format merger tree files [Srisawat et al., 2013]. See §12.37.2.

`mergerTreeImporterSussingASCII` Importer for "Sussing Merger Trees" ASCII format merger tree files [Srisawat et al., 2013].

`mergerTreeImporterSussingHDF5` Importer for "Sussing Merger Trees" HDF5 format merger tree files (Thomas et al.; in prep.).

`mergerTreeImporterGalacticus` Importer for Galacticus format merger tree files. See §12.37.1.

Merger Tree Building Mass Resolutions

Additional implementations for merger tree building mass resolutions are added using the `mergerTreeMassResolution` class. The implementation should be placed in a file containing the directive:

```
!# <mergerTreeMassResolution name="mergerTreeMassResolutionMyImplementation">
!# <description>A short description of the implementation.</description>
!# </mergerTreeMassResolution>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `mergerTreeMassResolutionMyImplementation`. The file *must* define a type that extends the `mergerTreeMassResolutionClass` class (or extends another type which is itself an extension of the `mergerTreeMassResolutionClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (mergerTreeMassResolutionClass), intent(inout)          :: self
  logical                               , intent(in   ), optional :: includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (mergerTreeMassResolutionClass), intent(inout)          :: self
 type   (inputParameters              ), intent(inout)          :: descriptor
 logical                               , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(mergerTreeMassResolutionClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (mergerTreeMassResolutionClass), intent(inout) :: self
 integer                               , intent(in   ) :: stateFile
 type   (fgsl_file                    ), intent(in   ) :: fgslStateFile
 integer(c_size_t                     ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (mergerTreeMassResolutionClass)                          , intent(inout) :: &
&  self
 type     (varying_string              ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                       )                          , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(mergerTreeMassResolutionClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`resolution` Gives the mass resolution to use for the given tree. Must have the following interface:

```
double precision function myImplementationResolution(self,tree)
 class(mergerTreeMassResolutionClass), intent(inout) :: self
 type (mergerTree                   ), intent(in   ) :: tree
end double precision function myImplementationResolution
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(mergerTreeMassResolutionClass), intent(inout) :: self
 class(mergerTreeMassResolutionClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (mergerTreeMassResolutionClass), intent(inout) :: self
 integer                               , intent(in   ) :: stateFile
 type   (fgsl_file                     ), intent(in   ) :: fgslStateFile
 integer(c_size_t                      ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`mergerTreeMassResolutionScaled` Provides a mass resolution for merger tree building which scales with tree mass. See §12.33.2.

`mergerTreeMassResolutionFixed` Provides a fixed mass resolution for merger tree building. See §12.33.1.

Merger Tree Node Evolvers

Additional implementations for merger tree node evolvers are added using the `mergerTreeNodeEvolver` class. The implementation should be placed in a file containing the directive:

```
!# <mergerTreeNodeEvolver name="mergerTreeNodeEvolverMyImplementation">
!# <description>A short description of the implementation.</description>
!# </mergerTreeNodeEvolver>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `mergerTreeNodeEvolverMyImplementation`. The file *must* define a type that extends the `mergerTreeNodeEvolverClass` class (or extends another type which is itself an extension of the `mergerTreeNodeEvolverClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`isAccurate` Return true if a tree node property is within expected accuracy of a given value. Must have the following interface:

```
 logical function myImplementationIsAccurate(self,valueNode, valueExpected)
  class           (mergerTreeNodeEvolverClass), intent(inout) :: self
  double precision                            , intent(in   ) :: valueNode, &
&  valueExpected
 end logical function myImplementationIsAccurate
```

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (mergerTreeNodeEvolverClass), intent(inout)           :: self
  logical                            , intent(in   ), optional :: includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationDescriptor(self,descriptor,includeMethod)
  class  (mergerTreeNodeEvolverClass), intent(inout)           :: self
  type   (inputParameters           ), intent(inout)           :: descriptor
  logical                            , intent(in   ), optional :: includeMethod
 end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAutoHook(self)
  class(mergerTreeNodeEvolverClass), intent(inout) :: self
 end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
  class  (mergerTreeNodeEvolverClass), intent(inout) :: self
  integer                            , intent(in   ) :: stateFile
  type   (fgsl_file                  ), intent(in   ) :: fgslStateFile
  integer(c_size_t                   ), intent(in   ) :: stateOperationID
 end subroutine myImplementationStateStore
```

`evolve` Evolve a node merger tree. Must have the following interface:

```
 subroutine myImplementationEvolve(self,tree,node,timeEnd,interrupted,functionInterrupt,&
&  galacticStructureSolver__,systemClockMaximum,status)
  class           (mergerTreeNodeEvolverClass  ), intent(inout)                    :: &
&  self
  type            (mergerTree                  ), intent(inout)                    :: &
&  tree
  type            (treeNode                    ), intent(inout)          , pointer :: &
&  node
  double precision                                , intent(in   )                    :: &
&  timeEnd
```

```
  logical                                          , intent(  out)                   :: &
&  interrupted
  procedure       (interruptTask               ), intent(  out)          , pointer :: &
&  functionInterrupt
  class           (galacticStructureSolverClass), intent(in   ), target            :: &
&  galacticStructureSolver__
  integer         (kind_int8                   ), intent(in   ), optional          :: &
&  systemClockMaximum
  integer                                          , intent(  out), optional          :: &
&  status
 end subroutine myImplementationEvolve
```

`promote` Promote `node` to its parent node, then destroy it. Must have the following interface:

```
subroutine myImplementationPromote(self,node)
 class(mergerTreeNodeEvolverClass), intent(inout)          :: self
 type (treeNode                  ), intent(inout), pointer :: node
end subroutine myImplementationPromote
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (mergerTreeNodeEvolverClass)                          , intent(inout) :: &
&  self
 type     (varying_string           ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                    )                          , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`merge` Handles instances where `node` is about to merge with its parent node. Must have the following interface:

```
subroutine myImplementationMerge(self,node)
 class(mergerTreeNodeEvolverClass), intent(inout)          :: self
 type (treeNode                  ), intent(inout), pointer :: node
end subroutine myImplementationMerge
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(mergerTreeNodeEvolverClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(mergerTreeNodeEvolverClass), intent(inout) :: self
 class(mergerTreeNodeEvolverClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (mergerTreeNodeEvolverClass), intent(inout) :: self
 integer                            , intent(in   ) :: stateFile
 type   (fgsl_file                 ), intent(in   ) :: fgslStateFile
 integer(c_size_t                  ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`mergerTreeNodeEvolverStandard` The standard merger tree noe evolver.

Merger Tree Node Merger Processing

Additional implementations for merger tree node merger processing are added using the `mergerTreeNodeMerger` class. The implementation should be placed in a file containing the directive:

```
!# <mergerTreeNodeMerger name="mergerTreeNodeMergerMyImplementation">
!# <description>A short description of the implementation.</description>
!# </mergerTreeNodeMerger>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `mergerTreeNode-MergerMyImplementation`. The file *must* define a type that extends the `mergerTreeNodeMergerClass` class (or extends another type which is itself an extension of the `mergerTreeNodeMergerClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
 class  (mergerTreeNodeMergerClass), intent(inout)          :: self
 logical                           , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (mergerTreeNodeMergerClass), intent(inout)          :: self
 type   (inputParameters          ), intent(inout)          :: descriptor
 logical                           , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(mergerTreeNodeMergerClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (mergerTreeNodeMergerClass), intent(inout) :: self
 integer                           , intent(in   ) :: stateFile
 type   (fgsl_file                ), intent(in   ) :: fgslStateFile
 integer(c_size_t                 ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (mergerTreeNodeMergerClass)                          , intent(inout) :: self
 type     (varying_string           ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                    )                          , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`process` Process the merger between `node` and its parent node, then destroy it. Must have the following interface:

```
subroutine myImplementationProcess(self,node)
 class(mergerTreeNodeMergerClass), intent(inout)          :: self
 type (treeNode                 ), intent(inout), pointer :: node
end subroutine myImplementationProcess
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(mergerTreeNodeMergerClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(mergerTreeNodeMergerClass), intent(inout) :: self
 class(mergerTreeNodeMergerClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (mergerTreeNodeMergerClass), intent(inout) :: self
 integer                           , intent(in   ) :: stateFile
 type   (fgsl_file                ), intent(in   ) :: fgslStateFile
 integer(c_size_t                 ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`mergerTreeNodeMergerSingleLevelHierarchy` A node merger class implementing a single level hierarchy.

Merger Tree Operators

Additional implementations for merger tree operators are added using the `mergerTreeOperator` class. The implementation should be placed in a file containing the directive:

```
!# <mergerTreeOperator name="mergerTreeOperatorMyImplementation">
!# <description>A short description of the implementation.</description>
!# </mergerTreeOperator>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `mergerTreeOperatorMyImplementation`. The file *must* define a type that extends the `mergerTreeOperatorClass` class (or extends another type which is itself an extension of the `mergerTreeOperatorClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`operate` Perform an operation on the merger tree. Must have the following interface:

```
subroutine myImplementationOperate(self,tree)
 class(mergerTreeOperatorClass), intent(inout)         :: self
 type (mergerTree             ), intent(inout), target :: tree
end subroutine myImplementationOperate
```

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
 class  (mergerTreeOperatorClass), intent(inout)           :: self
 logical                         , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (mergerTreeOperatorClass), intent(inout)           :: self
 type   (inputParameters        ), intent(inout)           :: descriptor
 logical                         , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(mergerTreeOperatorClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (mergerTreeOperatorClass), intent(inout) :: self
 integer                         , intent(in   ) :: stateFile
 type   (fgsl_file              ), intent(in   ) :: fgslStateFile
 integer(c_size_t               ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (mergerTreeOperatorClass)                          , intent(inout) :: self
 type     (varying_string         ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                  )                          , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(mergerTreeOperatorClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(mergerTreeOperatorClass), intent(inout) :: self
 class(mergerTreeOperatorClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (mergerTreeOperatorClass), intent(inout) :: self
 integer                         , intent(in   ) :: stateFile
 type   (fgsl_file              ), intent(in   ) :: fgslStateFile
 integer(c_size_t               ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

`finalize` Finalize a merger tree operator. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationFinalize(self)
 class(mergerTreeOperatorClass), intent(inout) :: self
end subroutine myImplementationFinalize
```

Existing implementations are:

`mergerTreeOperatorRegridTimes` Provides a merger tree operator which restructures the tree onto a fixed grid of timesteps.

`mergerTreeOperatorAugment` Provides a merger tree operator which augments tree resolution by inserting high-resolution branches.

`mergerTreeOperatorSelectWithinRange` Provides a select-within-range operator on the base nodes of merger trees.

`mergerTreeOperatorSequence` Provides a sequence of operators on merger trees.

`mergerTreeOperatorPruneClones` Provides a clone pruning operator on merger trees.

`mergerTreeOperatorMonotonizeMassGrowth` A merger tree operator which makes mass growth along merger tree branches monotonic.

`mergerTreeOperatorDeforest` Provides a deforestation operator for merger trees. Given a forest, this operator will destroy all but the first tree in the forest.

`mergerTreeOperatorPruneByTime` Provides a merger tree operator which prunes branches to end at a fixed time.

`mergerTreeOperatorPerturbMasses` A merger tree operator which perturbs halo masses by some error model.

`mergerTreeOperatorPruneLightcone` Provides a pruning-by-lightcone operator on merger trees. Trees which have no nodes which lie within the lightcone are completely pruned away.

`mergerTreeOperatorPruneBranchTips` Cmplements a merger tree operator which prunes tips of branches (i.e. sections from the leaf node to the first node with a sibling).

`mergerTreeOperatorPruneHierarchy` Contains a module which implements a merger tree operator which prunes branches below a given level in the substructure hierarchy. In any tree, the primary progenitor of the base node has substructure hierarchy depth 0. A branch which connects directly to this primary progenitor branch has substructure hierarchy depth 1, while a branch which connects directly to that branch has substructure hierarchy depth 2, and so on. The tree is pruned of all branches of hierarchy depth equal to or greater than the value provided by the `[hierarchyDepth]` parameter.

`mergerTreeOperatorPruneBaryons` Provides a pruning operator on merger trees that removes all branches that can not contain any baryons.

`mergerTreeOperatorProfiler` A merger tree operator which profiles merger tree structure.

`mergerTreeOperatorAssignOrbits` Provides a merger tree operator which assigns orbits to non-primary progenitor nodes.

`mergerTreeOperatorConditionalMF` Provides a merger tree operator which accumulates conditional mass functions for trees. In addition to the cumulative mass function, $1^{\rm st}$ through $n^{\rm th}$ most-massive progenitor mass functions, formation rate functions, and unevolved subhalo mass functions [Jiang and van den Bosch, 2014] split by hierarchy depth are computed and output. Mass functions are accumulated in logarithmically-spaced bins of parent halo mass, logarithmically-spaced bins of mass ratio (the ratio of progenitor to parent halo mass), and at pairs of parent/progenitor redshifts. The following parameters control the operator:

`parentMassCount` The number of bins in parent halo mass to use;

`parentMassMinimum` The minimum parent halo mass to consider;

`parentMassMaximum` The maximum parent halo mass to consider;

`massRatioCount` The number of bins in mass ratio to use;

`massRatioMinimum` The minimum mass ratio to consider;

`massRatioMaximum` The maximum mass ratio to consider;

`parentRedshifts` A list of redshifts at which to identify parent halos;

`progenitorRedshifts` A corresponding list of redshifts at which to identify progenitor halos;

`primaryProgenitorDepth` The number of $i^{\rm th}$ most-massive progenitor mass functions to compute (starting from the $1^{\rm st}$ most-massive);

`subhaloHierarchyDepth` The maximum depth in the subhalo hierarchy for which to compute the unevolved subhalo mass function;

`formationRateTimeFraction` The fraction of the current time over which to estimate the formation rate of halos when computing merger tree statistics;

`outputGroupName` The name of the hierarchical data format (HDF5) group to which mass functions will be written.

If the operator finds the named HDF5 group already in existance, it will accumulate its mass functions to those already written to the group, weighting by the inverse of the variance in each bin. The structure of the HDF5 group is as follows:

```
{
DATASET "conditionalMassFunction" {
COMMENT "Conditional mass functions []"
DATATYPE  H5T_IEEE_F64LE
DATASPACE  SIMPLE { ( Nratio, Nparent, Nz ) }
}
DATASET "conditionalMassFunctionError" {
COMMENT "Conditional mass function errors []"
DATATYPE  H5T_IEEE_F64LE
DATASPACE  SIMPLE { ( Nratio, Nparent, Nz ) }
}
DATASET "conditionalMassFunctionCovariance" {
COMMENT "Conditional mass function covariances []"
DATATYPE  H5T_IEEE_F64LE
DATASPACE  SIMPLE { ( Nratio, Nparent, Nz, Nratio, Nparent, Nz ) }
}
DATASET "formationRateFunction" {
COMMENT "Formation rate functions []"
DATATYPE  H5T_IEEE_F64LE
DATASPACE  SIMPLE { ( 2, Nratio, Nparent, Nz ) }
}
DATASET "formationRateFunctionError" {
COMMENT "Formation rate function errors []"
DATATYPE  H5T_IEEE_F64LE
DATASPACE  SIMPLE { ( 2, Nratio, Nparent, Nz ) }
}
DATASET "massParent" {
```

```
COMMENT "Mass of parent node [Msolar]"
DATATYPE  H5T_IEEE_F64LE
DATASPACE  SIMPLE { ( Nparent ) }
ATTRIBUTE "unitsInSI" {
DATATYPE  H5T_IEEE_F64LE
DATASPACE  SCALAR
DATA {
(0): 1.98892e+30
}
}
}
DATASET "massRatio" {
COMMENT "Mass of ratio node [Msolar]"
DATATYPE  H5T_IEEE_F64LE
DATASPACE  SIMPLE { ( Nratio ) }
ATTRIBUTE "unitsInSI" {
DATATYPE  H5T_IEEE_F64LE
DATASPACE  SCALAR
DATA {
(0): 1.98892e+30
}
}
}
DATASET "primaryProgenitorMassFunction" {
COMMENT "Primary progenitor mass functions []"
DATATYPE  H5T_IEEE_F64LE
DATASPACE  SIMPLE { ( Ndepth, Nratio, Nparent, Nz ) }
}
DATASET "primaryProgenitorMassFunctionError" {
COMMENT "Primary progenitor mass function errors []"
DATATYPE  H5T_IEEE_F64LE
DATASPACE  SIMPLE { ( Ndepth, Nratio, Nparent, Nz ) }
}
DATASET "redshiftParent" {
COMMENT "Redshift of parent node []"
DATATYPE  H5T_IEEE_F64LE
DATASPACE  SIMPLE { ( Nz ) }
}
DATASET "redshiftProgenitor" {
COMMENT "Redshift of progenitor node []"
DATATYPE  H5T_IEEE_F64LE
DATASPACE  SIMPLE { ( Nz ) }
}
DATASET "subhaloMassFunction" {
COMMENT "Unevolved subhalo mass functions []"
DATATYPE  H5T_IEEE_F64LE
DATASPACE  SIMPLE { ( Nhierarchy, Nratio, Nparent ) }
}
DATASET "subhaloMassFunctionError" {
```

```
COMMENT "Unevolved subhalo mass function errors []"
DATATYPE  H5T_IEEE_F64LE
DATASPACE  SIMPLE { ( Nhierarchy, Nratio, Nparent ) }
}
}
```

Where `Nratio` is the number of bins in mass ratio, `Nparent` is the number of bins in mass ratio, `Nz` is the number of parent/progenitor redshift pairs, `Ndepth` is the maximum depth in the ranking of most-massive progenitor mass functions, `Nhierarchy` is the maximum depth in the subhalo hierarchy for subhalo mass functions. The first dimension of the `formationRateFunction` dataset stores two different versions of the formation rate function. The first uses the mass of the forming halo at the time of formation, the second uses the mass of the node immediately prior to it becoming a subhalo. Mass functions are output as $\mathrm{d}N/\mathrm{d}\log_{10} m$ where N is the number of halos per parent halo, and m is the mass ratio.

`mergerTreeOperatorDumpToGraphViz` Provides a dump to GraphViz operator on merger trees.

`mergerTreeOperatorExport` A merger tree operator which exports merger trees to file.

`mergerTreeOperatorPruneNonEssential` Implements a merger tree operator which prunes branches that do not directly influence an "essential" node. Any branch which does not connect to the branch into which the node identified by ID `[essentialNodeI]` descends by time `essetialNodeTime]` will be pruned.

`mergerTreeOperatorParticulate` Provides a merger tree operator which create particle representations of Galacticus halos.

`mergerTreeOperatorInformationContent` A merger tree operator which computes the cladistic information content Thorley et al. [1998] of merger trees. This is output to a group in the output file with name specified by the `[outputGroupName]` parameter. Two datasets are written to this group: `treeIndex` which gives the index of each tree, and `informationContent` which gives the cladistic information content in units of bits.

`mergerTreeOperatorMassAccretionHistory` A merger tree operator which outputs mass accretion histories. Histories are written into the Galacticus output file in a group with name given by `[outputGroupName]`. Within that group, each merger tree has its own group named `mergerTree< N>` where `< N>` is the tree index. Within each such merger tree group datasets giving the node index ("`nodeIndex`"), time ("`nodeTime`"), basic mass ("`nodeMass`"), expansion factor ("`nodeExpansionFactor`") are written. Optionally, datasets giving the spin parameter ("`nodeSpin`") and its vector components ("`nodeSpinVector`") are included if `[includeSpin]` and `[includeSpinVector]` respectively are set to `true`.

`mergerTreeOperatorNull` Provides a null operator on merger trees.

`mergerTreeOperatorPruneByMass` Provides a pruning-by-mass operator on merger trees.

`mergerTreeOperatorOutputRootMasses` Output a file of tree root masses (and weights).

Merger Tree Outputters

Additional implementations for merger tree outputters are added using the `mergerTreeOutputter` class. The implementation should be placed in a file containing the directive:

```
!# <mergerTreeOutputter name="mergerTreeOutputterMyImplementation">
!# <description>A short description of the implementation.</description>
!# </mergerTreeOutputter>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `mergerTreeOutputterMyImplementation`. The file *must* define a type that extends the `mergerTreeOutputterClass` class (or extends another type which is itself an extension of the `mergerTreeOutputterClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
 class  (mergerTreeOutputterClass), intent(inout)          :: self
 logical                          , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (mergerTreeOutputterClass), intent(inout)          :: self
 type   (inputParameters         ), intent(inout)          :: descriptor
 logical                          , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(mergerTreeOutputterClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (mergerTreeOutputterClass), intent(inout) :: self
 integer                          , intent(in   ) :: stateFile
 type   (fgsl_file                ), intent(in   ) :: fgslStateFile
 integer(c_size_t                 ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (mergerTreeOutputterClass)                          , intent(inout) :: self
 type     (varying_string          ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                   )                          , intent(in   ) :: &
&  sourceName
end subroutine myImplementationAllowedParameters
```

`output` Output a merger tree. Must have the following interface:

```
subroutine myImplementationOutput(self,tree,indexOutput,time,isLastOutput)
 class           (mergerTreeOutputterClass), intent(inout)           :: self
 type            (mergerTree              ), intent(inout), target   :: tree
 integer         (c_size_t                ), intent(in   )           :: indexOutput
 double precision                           , intent(in   )           :: time
 logical                                    , intent(in   ), optional :: isLastOutput
end subroutine myImplementationOutput
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(mergerTreeOutputterClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(mergerTreeOutputterClass), intent(inout) :: self
 class(mergerTreeOutputterClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (mergerTreeOutputterClass), intent(inout) :: self
 integer                          , intent(in   ) :: stateFile
 type   (fgsl_file                ), intent(in   ) :: fgslStateFile
 integer(c_size_t                 ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

`reduce` Reduce the object onto another object of the class. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationReduce(self,reduced)
 class(mergerTreeOutputterClass), intent(inout) :: self
 class(mergerTreeOutputterClass), intent(inout) :: reduced
end subroutine myImplementationReduce
```

`finalize` Finalize output of merger trees. Must have the following interface:

```
subroutine myImplementationFinalize(self)
 class(mergerTreeOutputterClass), intent(inout) :: self
end subroutine myImplementationFinalize
```

Existing implementations are:

`mergerTreeOutputterAnalyzer` A merger tree outputter class which performs analyzes on the trees.

`mergerTreeOutputterNull` A merger tree outputter which does no output.

`mergerTreeOutputterStandard` The standard merger tree outputter.

`mergerTreeOutputterMulti` A merger tree outputter which combines multiple other outputters.

Merger Tree Walkers

Additional implementations for merger tree walkers are added using the `mergerTreeWalker` class. The implementation should be placed in a file containing the directive:

```
!# <mergerTreeWalker name="mergerTreeWalkerMyImplementation">
!# <description>A short description of the implementation.</description>
!# </mergerTreeWalker>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `mergerTreeWalkerMyImplementation`. The file *must* define a type that extends the `mergerTreeWalkerClass` class (or extends another type which is itself an extension of the `mergerTreeWalkerClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`next` Update the pointer to the next node to visit. Returns true if such a node exists, returns false if no such node exists (i.e. if all nodes have been visited already). Must have the following interface:

```
logical function myImplementationNext(self,node)
 class(mergerTreeWalkerClass), intent(inout)          :: self
 type (treeNode             ), intent(inout), pointer :: node
end logical function myImplementationNext
```

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
 class  (mergerTreeWalkerClass), intent(inout)           :: self
 logical                       , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (mergerTreeWalkerClass), intent(inout)           :: self
 type   (inputParameters      ), intent(inout)           :: descriptor
 logical                       , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(mergerTreeWalkerClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (mergerTreeWalkerClass), intent(inout) :: self
 integer                       , intent(in   ) :: stateFile
 type   (fgsl_file            ), intent(in   ) :: fgslStateFile
 integer(c_size_t             ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (mergerTreeWalkerClass)                          , intent(inout) :: self
 type     (varying_string       ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                )                          , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`nodesRemain` Returns true if more nodes remain to be walked to in the tree. Must have the following interface:

```
logical function myImplementationNodesRemain(self)
 class(mergerTreeWalkerClass), intent(inout) :: self
end logical function myImplementationNodesRemain
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(mergerTreeWalkerClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(mergerTreeWalkerClass), intent(inout) :: self
 class(mergerTreeWalkerClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (mergerTreeWalkerClass), intent(inout) :: self
 integer                       , intent(in   ) :: stateFile
 type   (fgsl_file            ), intent(in   ) :: fgslStateFile
 integer(c_size_t             ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`mergerTreeWalkerAllNodesBranch` Provides a merger tree walker which iterates depth-first over all all nodes in a given branch.

`mergerTreeWalkerAllNodes` Provides a merger tree walker which iterates depth-first over all all nodes.

`mergerTreeWalkerTreeConstruction` Provides a merger tree walker for trees under construction.

`mergerTreeWalkerAllAndFormationNodes` Provides a merger tree walker which iterates depth-first over all nodes including formation nodes.

`mergerTreeWalkerIsolatedNodesBranch` Provides a merger tree walker which iterates depth-first over all isolated nodes in a given branch.

`mergerTreeWalkerIsolatedNodes` Provides a merger tree walker which iterates depth-first over all isolated nodes.

Merger Tree Processing Times

Additional implementations for merger tree processing times are added using the `metaTreeProcessingTime` class. The implementation should be placed in a file containing the directive:

```
!# <metaTreeProcessingTime name="metaTreeProcessingTimeMyImplementation">
!# <description>A short description of the implementation.</description>
!# </metaTreeProcessingTime>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `metaTreeProcessingTimeMyImplementation`. The file *must* define a type that extends the `metaTreeProcessingTimeClass` class (or extends another type which is itself an extension of the `metaTreeProcessingTimeClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (metaTreeProcessingTimeClass), intent(inout)           :: self
  logical                             , intent(in   ), optional :: includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (metaTreeProcessingTimeClass), intent(inout)           :: self
 type   (inputParameters            ), intent(inout)           :: descriptor
 logical                             , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(metaTreeProcessingTimeClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`time` Return an estimate of the time needed to process a tree of the given mass. Must have the following interface:

```
double precision function myImplementationTime(self,massTree)
 class           (metaTreeProcessingTimeClass), intent(inout) :: self
 double precision                             , intent(in   ) :: massTree
end double precision function myImplementationTime
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (metaTreeProcessingTimeClass), intent(inout) :: self
 integer                             , intent(in   ) :: stateFile
 type   (fgsl_file                  ), intent(in   ) :: fgslStateFile
 integer(c_size_t                   ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (metaTreeProcessingTimeClass)                          , intent(inout) :: &
&  self
 type     (varying_string             ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                      )                          , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(metaTreeProcessingTimeClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(metaTreeProcessingTimeClass), intent(inout) :: self
 class(metaTreeProcessingTimeClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (metaTreeProcessingTimeClass), intent(inout) :: self
 integer                             , intent(in   ) :: stateFile
 type   (fgsl_file                  ), intent(in   ) :: fgslStateFile
 integer(c_size_t                   ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`metaTreeProcessingTimeFile` A merger tree processing time estimator using a polynomial relation read from file.

Model Parameters

Additional implementations for model parameters are added using the `modelParameter` class. The implementation should be placed in a file containing the directive:

```
!# <modelParameter name="modelParameterMyImplementation">
!# <description>A short description of the implementation.</description>
!# </modelParameter>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `modelParameterMyImplementation`. The file *must* define a type that extends the `modelParameterClass` class (or extends another type which is itself an extension of the `modelParameterClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`priorInvert` Invert the prior, returning the parameter value given the cumulative probability. Must have the following interface:

```
double precision function myImplementationPriorInvert(self,f)
 class           (modelParameterClass), intent(inout) :: self
 double precision                     , intent(in   ) :: f
end double precision function myImplementationPriorInvert
```

`priorMaximum` Return the maximum non-zero value of the prior for this parameter. Must have the following interface:

```
double precision function myImplementationPriorMaximum(self)
 class(modelParameterClass), intent(inout) :: self
end double precision function myImplementationPriorMaximum
```

`randomPerturbation` Return a random perturbation for this parameter. Must have the following interface:

```
double precision function myImplementationRandomPerturbation(self)
 class(modelParameterClass), intent(inout) :: self
end double precision function myImplementationRandomPerturbation
```

`priorSample` Sample from the parameter's prior. Must have the following interface:

```
double precision function myImplementationPriorSample(self)
 class(modelParameterClass), intent(inout) :: self
end double precision function myImplementationPriorSample
```

`map` Map the parameter value. Must have the following interface:

```
double precision function myImplementationMap(self,x)
 class           (modelParameterClass), intent(inout) :: self
 double precision                     , intent(in   ) :: x
end double precision function myImplementationMap
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(modelParameterClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`priorMinimum` Return the minimum non-zero value of the prior for this parameter. Must have the following interface:

```
double precision function myImplementationPriorMinimum(self)
 class(modelParameterClass), intent(inout) :: self
end double precision function myImplementationPriorMinimum
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (modelParameterClass), intent(inout)           :: self
 type   (inputParameters    ), intent(inout)           :: descriptor
 logical                     , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
 class  (modelParameterClass), intent(inout)           :: self
 logical                     , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(modelParameterClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`name` Return the name of this parameter. Must have the following interface:

```
type(varying_string) function myImplementationName(self)
 class(modelParameterClass), intent(inout) :: self
end type(varying_string) function myImplementationName
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (modelParameterClass), intent(inout) :: self
 integer                     , intent(in   ) :: stateFile
 type   (fgsl_file          ), intent(in   ) :: fgslStateFile
 integer(c_size_t           ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (modelParameterClass)                          , intent(inout) :: self
 type     (varying_string     ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*              )                          , intent(in   ) :: sourceName
end subroutine myImplementationAllowedParameters
```

`unmap` Unmap the parameter value. Must have the following interface:

```
double precision function myImplementationUnmap(self,x)
 class           (modelParameterClass), intent(inout) :: self
 double precision                     , intent(in   ) :: x
end double precision function myImplementationUnmap
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(modelParameterClass), intent(inout) :: self
 class(modelParameterClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (modelParameterClass), intent(inout) :: self
 integer                     , intent(in   ) :: stateFile
 type   (fgsl_file          ), intent(in   ) :: fgslStateFile
 integer(c_size_t           ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

`logPrior` Return the log-prior for this parameter. Must have the following interface:

```
double precision function myImplementationLogPrior(self,x)
 class           (modelParameterClass), intent(inout) :: self
 double precision                     , intent(in   ) :: x
end double precision function myImplementationLogPrior
```

Existing implementations are:

`modelParameterInactive` An inactive model parameter class.

`modelParameterDerived` A model parameter class in which the parameter value is derived from other parameters.

`modelParameterActive` An active model parameter class.

N-body Halo Mass Errors

Additional implementations for n-body halo mass errors are added using the `nbodyHaloMassError` class. The implementation should be placed in a file containing the directive:

```
!# <nbodyHaloMassError name="nbodyHaloMassErrorMyImplementation">
!# <description>A short description of the implementation.</description>
!# </nbodyHaloMassError>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `nbodyHaloMassErrorMyImplementation`. The file *must* define a type that extends the `nbodyHaloMassErrorClass` class (or extends another type which is itself an extension of the `nbodyHaloMassErrorClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
 class  (nbodyHaloMassErrorClass), intent(inout)           :: self
 logical                         , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (nbodyHaloMassErrorClass), intent(inout)           :: self
 type   (inputParameters        ), intent(inout)           :: descriptor
 logical                         , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(nbodyHaloMassErrorClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (nbodyHaloMassErrorClass), intent(inout) :: self
 integer                         , intent(in   ) :: stateFile
 type   (fgsl_file               ), intent(in   ) :: fgslStateFile
 integer(c_size_t                ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (nbodyHaloMassErrorClass)                          , intent(inout) :: self
 type     (varying_string         ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                  )                          , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`errorFractional` Return the fractional error on the mass of an N-body halo corresponding to the given `node`. Must have the following interface:

```
double precision function myImplementationErrorFractional(self,node)
 class(nbodyHaloMassErrorClass), intent(inout) :: self
 type (treeNode               ), intent(inout) :: node
end double precision function myImplementationErrorFractional
```

`correlation` Return the correlation in the error on the mass of a pair of N-body halos corresponding to the given `node1` and `node2`. Must have the following interface:

```
double precision function myImplementationCorrelation(self,node1, node2)
 class(nbodyHaloMassErrorClass), intent(inout) :: self
 type (treeNode               ), intent(inout) :: node1, node2
end double precision function myImplementationCorrelation
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(nbodyHaloMassErrorClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(nbodyHaloMassErrorClass), intent(inout) :: self
 class(nbodyHaloMassErrorClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`errorZeroAlways` Return `true` if the mass error is always zero for any halo. A default implementation exists. If overridden the following interface must be used:

```
logical function myImplementationErrorZeroAlways(self)
 class(nbodyHaloMassErrorClass), intent(inout) :: self
end logical function myImplementationErrorZeroAlways
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (nbodyHaloMassErrorClass), intent(inout) :: self
 integer                         , intent(in   ) :: stateFile
 type   (fgsl_file              ), intent(in   ) :: fgslStateFile
 integer(c_size_t               ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`nbodyHaloMassErrorTrenti2010` An N-body dark matter halo mass error class using the model of Trenti et al. [2010].

`nbodyHaloMassErrorNull` A null N-body dark matter halo mass error class. Errors are always zero.

`nbodyHaloMassErrorSOHaloFinder` An N-body dark matter halo mass error class which implements a model for errors in spherical overdensity halo finders.

`nbodyHaloMassErrorPowerLaw` An N-body dark matter halo mass error class in which errors are a power-law in halo mass.

`nbodyHaloMassErrorFriendsOfFriends` An N-body dark matter halo mass error class which uses a fit appropriate for friends-of-friends group finders.

N-Body Simulation Data Importer

Additional implementations for n-body simulation data importer are added using the `nbodyImporter` class. The implementation should be placed in a file containing the directive:

```
!# <nbodyImporter name="nbodyImporterMyImplementation">
!# <description>A short description of the implementation.</description>
!# </nbodyImporter>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `nbodyImporterMyImplementation`. The file *must* define a type that extends the `nbodyImporterClass` class (or extends another type which is itself an extension of the `nbodyImporterClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (nbodyImporterClass), intent(inout)           :: self
  logical                    , intent(in   ), optional :: includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (nbodyImporterClass), intent(inout)           :: self
 type   (inputParameters   ), intent(inout)           :: descriptor
 logical                    , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(nbodyImporterClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (nbodyImporterClass), intent(inout) :: self
 integer                    , intent(in   ) :: stateFile
 type   (fgsl_file         ), intent(in   ) :: fgslStateFile
 integer(c_size_t          ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (nbodyImporterClass)                          , intent(inout) :: self
 type     (varying_string    ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*             )                          , intent(in   ) :: sourceName
end subroutine myImplementationAllowedParameters
```

`import` Import position and velocity data from the named N-body data file. Must have the following interface:

```
type(nBodyData) function myImplementationImport(self,fileName,fileNamePrevious)
 class    (nbodyImporterClass), intent(inout)           :: self
 character(len=*             ), intent(in   )           :: fileName
 character(len=*             ), intent(in   ), optional :: fileNamePrevious
end type(nBodyData) function myImplementationImport
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(nbodyImporterClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(nbodyImporterClass), intent(inout) :: self
 class(nbodyImporterClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (nbodyImporterClass), intent(inout) :: self
 integer                    , intent(in   ) :: stateFile
 type   (fgsl_file         ), intent(in   ) :: fgslStateFile
 integer(c_size_t          ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`nbodyImporterGadgetBinary` An importer for Gadget binary files.

`nbodyImporterGadgetHDF5` An importer for Gadget HDF5 files.

N-Body Simulation Data Operators

Additional implementations for n-body simulation data operators are added using the `nbodyOperator` class. The implementation should be placed in a file containing the directive:

```
!# <nbodyOperator name="nbodyOperatorMyImplementation">
!# <description>A short description of the implementation.</description>
!# </nbodyOperator>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `nbodyOperatorMyImplementation`. The file *must* define a type that extends the `nbodyOperatorClass` class (or extends another type which is itself an extension of the `nbodyOperatorClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`operate` Operate on the provided N-body simulation. Must have the following interface:

```
subroutine myImplementationOperate(self,simulation)
 class(nbodyOperatorClass), intent(inout) :: self
 type (nBodyData         ), intent(inout) :: simulation
end subroutine myImplementationOperate
```

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (nbodyOperatorClass), intent(inout)           :: self
  logical                    , intent(in   ), optional :: includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (nbodyOperatorClass), intent(inout)           :: self
 type   (inputParameters   ), intent(inout)           :: descriptor
 logical                    , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(nbodyOperatorClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (nbodyOperatorClass), intent(inout) :: self
 integer                    , intent(in   ) :: stateFile
 type   (fgsl_file         ), intent(in   ) :: fgslStateFile
 integer(c_size_t          ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (nbodyOperatorClass)                          , intent(inout) :: self
 type     (varying_string    ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*             )                          , intent(in   ) :: sourceName
end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(nbodyOperatorClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(nbodyOperatorClass), intent(inout) :: self
 class(nbodyOperatorClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (nbodyOperatorClass), intent(inout) :: self
 integer                    , intent(in   ) :: stateFile
 type   (fgsl_file         ), intent(in   ) :: fgslStateFile
 integer(c_size_t          ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`nbodyOperatorNull` A null N-body data operator.

`nbodyOperatorEnvironmentalOverdensity` An N-body data operator which determines the environmental overoverdensity around particles.

`nbodyOperatorMeanPosition` An N-body data operator which determines the mean position and velocity of particles.

`nbodyOperatorRotationCurve` An N-body data operator which computes the rotation curve at a set of given radii.

`nbodyOperatorSequence` An N-body data operator which applies a sequence of other operators.

`nbodyOperatorSelfBound` An N-body data operator which determines the subset of particles that are self-bound.

`nbodyOperatorPairCounts` An N-body data operator which computes pair counts in bins of separation.

`nbodyOperatorVelocityDispersion` An N-body data operator which computes the rotation curve at a set of given radii.

Output Analysis Property Extractor

Additional implementations for output analysis property extractor are added using the `nodePropertyExtractor` class. The implementation should be placed in a file containing the directive:

```
!# <nodePropertyExtractor name="nodePropertyExtractorMyImplementation">
!# <description>A short description of the implementation.</description>
!# </nodePropertyExtractor>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `nodePropertyExtractorMyImplementation`. The file *must* define a type that extends the `nodePropertyExtractorClass` class (or extends another type which is itself an extension of the `nodePropertyExtractorClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
 class  (nodePropertyExtractorClass), intent(inout)           :: self
 logical                            , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (nodePropertyExtractorClass), intent(inout)           :: self
 type   (inputParameters           ), intent(inout)           :: descriptor
 logical                            , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(nodePropertyExtractorClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`addInstances` Add multiple instances of this property to a `multiCounter` object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAddInstances(self,node,instance)
 class(nodePropertyExtractorClass), intent(inout) :: self
 type (treeNode                  ), intent(inout) :: node
 type (multiCounter              ), intent(inout) :: instance
end subroutine myImplementationAddInstances
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (nodePropertyExtractorClass), intent(inout) :: self
 integer                            , intent(in   ) :: stateFile
 type   (fgsl_file                  ), intent(in   ) :: fgslStateFile
 integer(c_size_t                   ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`quantity` Return the class of the extracted property. A default implementation exists. If overridden the following interface must be used:

```
integer function myImplementationQuantity(self)
 class(nodePropertyExtractorClass), intent(inout) :: self
end integer function myImplementationQuantity
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (nodePropertyExtractorClass)                          , intent(inout) :: &
&  self
 type     (varying_string           ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                     )                          , intent(in   ) :: &
&  sourceName
end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(nodePropertyExtractorClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(nodePropertyExtractorClass), intent(inout) :: self
 class(nodePropertyExtractorClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`type` Return the type of the extracted property. Must have the following interface:

```
integer function myImplementationType(self)
 class(nodePropertyExtractorClass), intent(inout) :: self
end integer function myImplementationType
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (nodePropertyExtractorClass), intent(inout) :: self
 integer                            , intent(in   ) :: stateFile
 type   (fgsl_file                 ), intent(in   ) :: fgslStateFile
 integer(c_size_t                  ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`nodePropertyExtractorVirialProperties` A property extractor class for virial radius and velocity.

`nodePropertyExtractorMassHalo` A halo mass output analysis property extractor class.

`nodePropertyExtractorDescendents` An ISM mass output analysis property extractor class.

`nodePropertyExtractorMassProfile` A property extractor class for the mass enclosed by a set of radii.

`nodePropertyExtractorSatelliteOrbitalExtrema` A satellite orbital extrema luminosity property extractor class.

`nodePropertyExtractorDensityContrasts` A property extractor class for the mass and radii of spheres are specified density contrast.

`nodePropertyExtractorSpinBullock` A node property extractor class for the Bullock et al. [2001] definition of spin parameter.

`nodePropertyExtractorRotationCurve` A property extractor class for the rotation curve at a set of radii.

`nodePropertyExtractorRadiusCooling` A cooling radius property extractor class.

`nodePropertyExtractorIndicesHost` An ISM mass output analysis property extractor class.

`nodePropertyExtractorHaloBias` A node property extractor class for halo bias.

`nodePropertyExtractorICMSZ` An intracluster medium Sunyaev-Zeldovich property extractor class.

`nodePropertyExtractorMassStellar` A stellar mass output analysis property extractor class.

`nodePropertyExtractorMainBranchStatus` An ISM mass output analysis property extractor class.

`nodePropertyExtractorNodeIndices` A property extractor class for basic node indices.

`nodePropertyExtractorNull` A null output analysis property extractor class.

`nodePropertyExtractorRatio` A ratio output analysis property extractor class.

`nodePropertyExtractorMassISM` An ISM mass output analysis property extractor class.

`nodePropertyExtractorIntegerTuple` An abstract output analysis property extractor class which provides a tuple of integer properties.

`nodePropertyExtractorScalar` An abstract output analysis property extractor class which provieds a scalar floating point property.

`nodePropertyExtractorRadiusHalfMass` A radiusHalfMass property extractor class.

`nodePropertyExtractorMassBlackHole` An ISM mass output analysis property extractor class.

`nodePropertyExtractorIntegerScalar` An abstract output analysis property extractor class which provieds a scalar integer property.

`nodePropertyExtractorLmnstyStllrCF2000` A stellar luminosity output analysis property extractor class which applies the dust model of Charlot and Fall [2000].

`nodePropertyExtractorVelocityMaximum` A cooling rate property extractor class.

`nodePropertyExtractorMetallicityISM` An ISM metallicity output analysis property extractor class.

`nodePropertyExtractorMassStellarSpheroid` A spheroid stellar mass output analysis property extractor class.

`nodePropertyExtractorRadiiHalfLightProperties` A half-light radii property extractor class.

`nodePropertyExtractorConcentration` A concentration output analysis property extractor class.

`nodePropertyExtractorProjectedDensity` A property extractor class for the projected density at a set of radii.

`nodePropertyExtractorMostMassiveProgenitor` An ISM mass output analysis property extractor class.

`nodePropertyExtractorFinalDescendent` An ISM mass output analysis property extractor class.

`nodePropertyExtractorSatelliteStatus` An ISM mass output analysis property extractor class.

`nodePropertyExtractorHalfMassRadius` A half-(stellar) mass output analysis property extractor class.

`nodePropertyExtractorMassStellarMorphology` A stellar mass-weighted morphology output analysis property extractor class.

`nodePropertyExtractorTreeWeight` A merger tree weight property extractor class.

`nodePropertyExtractorSpin` A spin parameter output analysis property extractor class.

`nodePropertyExtractorRedshiftLastIsolated` A redshiftLastIsolated property extractor class.

`nodePropertyExtractorDensityProfile` A property extractor class for the density at a set of radii.

`nodePropertyExtractorMulti` A multi output extractor property extractor class.

`nodePropertyExtractorLightcone` A lightcone output extractor property extractor class.

`nodePropertyExtractorIndicesTree` An ISM mass output analysis property extractor class.

`nodePropertyExtractorMassHost` A host halo mass property extractor class.

`nodePropertyExtractorHaloEnvironment` A node property extractor class for halo environment.

`nodePropertyExtractorRateCooling` A cooling rate property extractor class.

`nodePropertyExtractorFractionAccretionHotMode` A hot mode accretion fraction property extractor class.

`nodePropertyExtractorICMXRayLuminosity` An intracluster medium X-ray luminosity property extractor class.

`nodePropertyExtractorLmnstyEmssnLine` A stellar luminosity output analysis property extractor class.

`nodePropertyExtractorLuminosityStellar` A stellar luminosity output analysis property extractor class.

`nodePropertyExtractorVelocityDispersion` A property extractor class for the velocity dispersion at a set of radii.

`nodePropertyExtractorTuple` An abstract output analysis property extractor class which provieds a tuple of floating point properties.

`nodePropertyExtractorRateInfallColdMode` A cold mode infall rate property extractor class.

Unary Operators

Additional implementations for unary operators are added using the `operatorUnary` class. The implementation should be placed in a file containing the directive:

```
!# <operatorUnary name="operatorUnaryMyImplementation">
!# <description>A short description of the implementation.</description>
!# </operatorUnary>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `operator-UnaryMyImplementation`. The file *must* define a type that extends the `operatorUnaryClass` class (or extends another type which is itself an extension of the `operatorUnaryClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`operate` Operate on the given value. Must have the following interface:

```
double precision function myImplementationOperate(self,x)
 class           (operatorUnaryClass), intent(inout) :: self
 double precision                    , intent(in   ) :: x
end double precision function myImplementationOperate
```

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
 class  (operatorUnaryClass), intent(inout)           :: self
 logical                    , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (operatorUnaryClass), intent(inout)           :: self
 type   (inputParameters   ), intent(inout)           :: descriptor
 logical                    , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(operatorUnaryClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (operatorUnaryClass), intent(inout) :: self
 integer                    , intent(in   ) :: stateFile
 type   (fgsl_file         ), intent(in   ) :: fgslStateFile
 integer(c_size_t          ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (operatorUnaryClass)                          , intent(inout) :: self
 type     (varying_string    ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*             )                          , intent(in   ) :: sourceName
end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(operatorUnaryClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(operatorUnaryClass), intent(inout) :: self
 class(operatorUnaryClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`unoperate` Reverse the operation. Must have the following interface:

```
double precision function myImplementationUnoperate(self,f)
 class           (operatorUnaryClass), intent(inout) :: self
 double precision                    , intent(in   ) :: f
end double precision function myImplementationUnoperate
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (operatorUnaryClass), intent(inout) :: self
 integer                    , intent(in   ) :: stateFile
 type   (fgsl_file         ), intent(in   ) :: fgslStateFile
 integer(c_size_t          ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`operatorUnaryInverse` An inverse unary operator.

`operatorUnaryIdentity` An identity unary operator.

`operatorUnaryLogarithm` A logarithm unary operator.

Output Analysis

Additional implementations for output analysis are added using the `outputAnalysis` class. The implementation should be placed in a file containing the directive:

```
!# <outputAnalysis name="outputAnalysisMyImplementation">
!# <description>A short description of the implementation.</description>
!# </outputAnalysis>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `outputAnalysisMyImplementation`. The file *must* define a type that extends the `outputAnalysisClass` class (or extends another type which is itself an extension of the `outputAnalysisClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
 class  (outputAnalysisClass), intent(inout)           :: self
 logical                     , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (outputAnalysisClass), intent(inout)           :: self
 type   (inputParameters    ), intent(inout)           :: descriptor
 logical                     , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(outputAnalysisClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (outputAnalysisClass), intent(inout) :: self
 integer                     , intent(in   ) :: stateFile
 type   (fgsl_file          ), intent(in   ) :: fgslStateFile
 integer(c_size_t           ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (outputAnalysisClass)                          , intent(inout) :: self
 type     (varying_string     ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*              )                          , intent(in   ) :: sourceName
end subroutine myImplementationAllowedParameters
```

`analyze` Analyze the given `node`. Must have the following interface:

```
subroutine myImplementationAnalyze(self,node,iOutput)
 class  (outputAnalysisClass), intent(inout) :: self
 type   (treeNode           ), intent(inout) :: node
 integer(c_size_t           ), intent(in   ) :: iOutput
end subroutine myImplementationAnalyze
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(outputAnalysisClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(outputAnalysisClass), intent(inout) :: self
 class(outputAnalysisClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (outputAnalysisClass), intent(inout) :: self
 integer                     , intent(in   ) :: stateFile
 type   (fgsl_file          ), intent(in   ) :: fgslStateFile
 integer(c_size_t           ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

`logLikelihood` Return the log-likelihood of the analysis. Must have the following interface:

```
double precision function myImplementationLogLikelihood(self)
 class(outputAnalysisClass), intent(inout) :: self
end double precision function myImplementationLogLikelihood
```

`reduce` Reduce the object onto another object of the class. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationReduce(self,reduced)
 class(outputAnalysisClass), intent(inout) :: self
 class(outputAnalysisClass), intent(inout) :: reduced
end subroutine myImplementationReduce
```

`finalize` Finalize the analysis. Must have the following interface:

```
subroutine myImplementationFinalize(self)
 class(outputAnalysisClass), intent(inout) :: self
end subroutine myImplementationFinalize
```

Existing implementations are:

`outputAnalysisMassFunctionStellarBernardi2013SDSS` A Bernardi et al. [2013] stellar mass function output analysis class.

`outputAnalysisMassFunctionHIALFALFAMartin2010` An ALFALFA HI mass function output analysis class.

`outputAnalysisMassFunctionStellarBaldry2012GAMA` A Baldry et al. [2012] stellar mass function output analysis class.

`outputAnalysisLocalGroupMassFunction` An output analysis class for Local Group satellite galaxy mass functions.

`outputAnalysisMassFunctionStellarVIPERS` A VIPERS stellar mass function output analysis class.

`outputAnalysisHIVsHaloMassRelationPadmanabhan2017` An HI vs halo mass relation output analysis class.

`outputAnalysisVolumeFunction1D` A generic 1D volume function (i.e. number density of objects binned by some property, e.g. a mass function) output analysis class.

`outputAnalysisConcentrationVsHaloMassCDMLudlow2016` A concentration vs. halo mass analysis class matched to the Ludlow et al. [2016] CDM sample..

`outputAnalysisMeanFunction1D` A generic 1D mean function (i.e. mean value of some property weighted by number density of objects binned by some property) output analysis class.

`outputAnalysisMassFunctionStellar` A stellar mass function output analysis class.

`outputAnalysisColorDistributionSDSS` An SDSS color distribution function output analysis class.

`outputAnalysisConcentrationDistributionCDMCOCO` A concentration distribution function output analysis class for COCO CDM data.

`outputAnalysisLuminosityFunctionSobral2013HiZELS` An SDSS Hα luminosity function output analysis class for the Sobral et al. [2013] analysis.

`outputAnalysisLuminosityFunctionMonteroDorta2009SDSS` An SDSS luminosity function output analysis class for the Montero-Dorta and Prada [2009] analysis.

`outputAnalysisMassFunctionStellarULTRAVISTA` An ULTRAVISTA stellar mass function output analysis class.

`outputAnalysisLuminosityFunctionHalpha` A luminosity function output analysis class.

`outputAnalysisMorphologicalFractionGAMAMoffett2016` A morphological fraction output analysis class for the analysis of Moffett et al. [2016].

`outputAnalysisCorrelationFunctionHearin2013SDSS` A correlation function output analysis class for the Hearin et al. [2013] analysis.

`outputAnalysisCorrelationFunction` A generic two-point correlation function output analysis class.

`outputAnalysisSpinDistributionBett2007` A stellar mass function output analysis class.

`outputAnalysisMassFunctionStellarPRIMUS` A PRIMUS stellar mass function output analysis class.

`outputAnalysisMassFunctionStellarUKIDSSUDS` A UKIDSS UDS stellar mass function output analysis class.

`outputAnalysisMassMetallicityAndrews2013` A mass-metallicity relation output analysis class.

`outputAnalysisScatterFunction1D` A generic 1D scatter function (i.e. the scatter of some property weighted by number density of objects binned by some property) output analysis class.

`outputAnalysisLuminosityFunction` A luminosity function output analysis class.

`outputAnalysisNull` A null output analysis class.

`outputAnalysisMassMetallicityBlanc2017` A mass-metallicity relation output analysis class.

`outputAnalysisStellarVsHaloMassRelationLeauthaud2012` A stellar vs halo mass relation output analysis class.

`outputAnalysisLuminosityFunctionGunawardhana2013SDSS` An SDSS Hα luminosity function output analysis class for the Gunawardhana et al. [2013] analysis.

`outputAnalysisMassFunctionHI` An HI mass function output analysis class.

`outputAnalysisMassFunctionStellarZFOURGE` A ZFOURGE stellar mass function output analysis class.

`outputAnalysisGalaxySizesSDSS` A stellar mass function output analysis class.

`outputAnalysisBlackHoleBulgeRelation` A mass-metallicity relation output analysis class.

`outputAnalysisMassFunctionStellarSDSS` An SDSS stellar mass function output analysis class.

`outputAnalysisMulti` A merger tree analysis class which combines multiple other analyses.

Output Analysis Distribution Normalizer

Additional implementations for output analysis distribution normalizer are added using the `outputAnalysisDistributionNor` class. The implementation should be placed in a file containing the directive:

```
!# <outputAnalysisDistributionNormalizer name="outputAnalysisDistributionNormalizerMyImplementation">
!# <description>A short description of the implementation.</description>
!# </outputAnalysisDistributionNormalizer>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `outputAnalysisDistributionNormalizerMyImplementation`. The file *must* define a type that extends the `outputAnalysisDistributionNormalizerClass` class (or extends another type which is itself an extension of the `outputAnalysisDistributionNormalizerClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (outputAnalysisDistributionNormalizerClass), intent(inout)          :: self
  logical                                           , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationDescriptor(self,descriptor,includeMethod)
  class  (outputAnalysisDistributionNormalizerClass), intent(inout)          :: self
  type   (inputParameters                          ), intent(inout)          :: &
&  descriptor
  logical                                           , intent(in   ), optional :: &
&  includeMethod
 end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAutoHook(self)
  class(outputAnalysisDistributionNormalizerClass), intent(inout) :: self
 end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
  class  (outputAnalysisDistributionNormalizerClass), intent(inout) :: self
  integer                                           , intent(in   ) :: stateFile
  type   (fgsl_file                                 ), intent(in   ) :: fgslStateFile
  integer(c_size_t                                  ), intent(in   ) :: stateOperationID
 end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
  class    (outputAnalysisDistributionNormalizerClass)                          , &
&  intent(inout) :: self
  type     (varying_string                            ), allocatable, dimension(:), &
&  intent(inout) :: allowedParameters
  character(len=*                                     )                          , &
&  intent(in   ) :: sourceName
 end subroutine myImplementationAllowedParameters
```

`normalize` Normalize a distribution. Must have the following interface:

```
 subroutine myImplementationNormalize(self,distribution,covariance,propertyValueMinimum,&
&   propertyValueMaximum)
  class           (outputAnalysisDistributionNormalizerClass)               , intent(&
&  inout) :: self
  double precision                                          , dimension(:)  , intent(&
&  inout) :: distribution
```

```
  double precision                                          , dimension(:,:), intent(&
&  inout) :: covariance
  double precision                                          , dimension(:)  , intent(in&
&     ) :: propertyValueMinimum, propertyValueMaximum
 end subroutine myImplementationNormalize
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(outputAnalysisDistributionNormalizerClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(outputAnalysisDistributionNormalizerClass), intent(inout) :: self
 class(outputAnalysisDistributionNormalizerClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (outputAnalysisDistributionNormalizerClass), intent(inout) :: self
 integer                                           , intent(in   ) :: stateFile
 type   (fgsl_file                                 ), intent(in   ) :: fgslStateFile
 integer(c_size_t                                  ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`outputAnalysisDistributionNormalizerSequence` Provides a sequence of normalizers on on-the-fly outputs.

`outputAnalysisDistributionNormalizerIdentity` An identity output analysis distribution normalizer class.

`outputAnalysisDistributionNormalizerBinWidth` A bin width output analysis distribution normalizer class.

`outputAnalysisDistributionNormalizerUnitarity` A unitarity output analysis distribution normalizer class.

`outputAnalysisDistributionNormalizerLog10ToLog` A $\log_{10} \rightarrow$ log output analysis distribution normalizer class.

Output Analysis Distribution Operator

Additional implementations for output analysis distribution operator are added using the `outputAnalysisDistributionOpera` class. The implementation should be placed in a file containing the directive:

```
!# <outputAnalysisDistributionOperator name="outputAnalysisDistributionOperatorMyImplementation">
!# <description>A short description of the implementation.</description>
!# </outputAnalysisDistributionOperator>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `outputAnalysisDistributionOperatorMyImplementation`. The file *must* define a type that extends the `outputAnalysisDistributionOperatorClass` class (or extends another type which is itself an extension of the `outputAnalysisDistributionOperatorClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (outputAnalysisDistributionOperatorClass), intent(inout)           :: self
  logical                                         , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationDescriptor(self,descriptor,includeMethod)
  class  (outputAnalysisDistributionOperatorClass), intent(inout)           :: self
  type   (inputParameters                        ), intent(inout)           :: &
&  descriptor
  logical                                         , intent(in   ), optional :: &
&  includeMethod
 end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAutoHook(self)
  class(outputAnalysisDistributionOperatorClass), intent(inout) :: self
 end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
  class  (outputAnalysisDistributionOperatorClass), intent(inout) :: self
  integer                                         , intent(in   ) :: stateFile
  type   (fgsl_file                               ), intent(in   ) :: fgslStateFile
  integer(c_size_t                                ), intent(in   ) :: stateOperationID
 end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
  class    (outputAnalysisDistributionOperatorClass)                          , intent(&
&  inout) :: self
```

```
  type     (varying_string                          ), allocatable, dimension(:), intent(&
&  inout) :: allowedParameters
  character(len=*                                   )                           , intent(&
&  in   ) :: sourceName
 end subroutine myImplementationAllowedParameters
```

`operateScalar` Operate on a scalar to produce a distribution. Must have the following interface:

```
 double precision, dimension(size(propertyValueMinimum)) function &
&  myImplementationOperateScalar(self,propertyValue,propertyType,propertyValueMinimum, &
&  propertyValueMaximum,outputIndex,node)
  class           (outputAnalysisDistributionOperatorClass)              , intent(inout)&
&   :: self
  double precision                                                       , intent(in   )&
&   :: propertyValue
  integer                                                                , intent(in   )&
&   :: propertyType
  double precision                                          , dimension(:), intent(in   )&
&   :: propertyValueMinimum, propertyValueMaximum
  integer         (c_size_t                                )             , intent(in   )&
&   :: outputIndex
  type            (treeNode                                )             , intent(inout)&
&   :: node
 end double precision, dimension(size(propertyValueMinimum)) function &
&  myImplementationOperateScalar
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(outputAnalysisDistributionOperatorClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(outputAnalysisDistributionOperatorClass), intent(inout) :: self
 class(outputAnalysisDistributionOperatorClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (outputAnalysisDistributionOperatorClass), intent(inout) :: self
 integer                                         , intent(in   ) :: stateFile
 type   (fgsl_file                               ), intent(in   ) :: fgslStateFile
 integer(c_size_t                                ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

`operateDistribution` Operate on a distribution to produce a distribution. Must have the following interface:

```
 double precision, dimension(size(propertyValueMinimum)) function &
&  myImplementationOperateDistribution(self,distribution,propertyType,&
&  propertyValueMinimum, propertyValueMaximum,outputIndex,node)
  class           (outputAnalysisDistributionOperatorClass)             , intent(inout)&
&   :: self
  double precision                                                , dimension(:), intent(in   )&
&   :: distribution
  integer                                                                       , intent(in   )&
&   :: propertyType
  double precision                                                , dimension(:), intent(in   )&
&   :: propertyValueMinimum, propertyValueMaximum
  integer         (c_size_t                                      )              , intent(in   )&
&   :: outputIndex
  type            (treeNode                                      )              , intent(inout)&
&   :: node
 end double precision, dimension(size(propertyValueMinimum)) function &
&  myImplementationOperateDistribution
```

Existing implementations are:

`outputAnalysisDistributionOperatorRndmErrNbodyMass` A random error output analysis distribution operator class providing errors in $\log_{10}$ of N-body halo mass.

`outputAnalysisDistributionOperatorIdentity` A identity output analysis distribution operator class.

`outputAnalysisDistributionOperatorGrvtnlLnsng` A gravitational lensing output analysis distribution operator class.

`outputAnalysisDistributionOperatorRandomErrorALFLF` A random error output analysis distribution operator class providing errors in HI mass for the ALFALFA survey. Specifically, $\sigma_{\rm obs} = a + \exp\left(-\frac{\log_{10}(M_{\rm HI}/M_\odot)-b}{c}\right)$.

`outputAnalysisDistributionOperatorRandomErrorFixed` A random error output analysis distribution operator class.

`outputAnalysisDistributionOperatorRandomErrorPlynml` A random error output analysis distribution operator class.

`outputAnalysisDistributionOperatorSpinNBodyErrors` An output analysis distribution operator class to account for errors on N-body measurements of halo spin.

`outputAnalysisDistributionOperatorSequence` A sequence output analysis distribution operator class.

`outputAnalysisDistributionOperatorRandomError` A random error output analysis distribution operator class.

`outputAnalysisDistributionOperatorRndmErrNbdyCnc` A random error output analysis distribution operator class providing errors in $\log_{10}$ of N-body halo concentration.

`outputAnalysisDistributionOperatorDiskSizeInclntn` An output analysis distribution operator class which implements the effects of inclination on disk size.

Output Analysis Molecular Ratio

Additional implementations for output analysis molecular ratio are added using the `outputAnalysisMolecularRatio` class. The implementation should be placed in a file containing the directive:

```
!# <outputAnalysisMolecularRatio name="outputAnalysisMolecularRatioMyImplementation">
!# <description>A short description of the implementation.</description>
!# </outputAnalysisMolecularRatio>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `outputAnalysisMolecularRatioMyImplementation`. The file *must* define a type that extends the `outputAnalysisMolecularRatioClass` class (or extends another type which is itself an extension of the `outputAnalysisMolecularRatioClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`ratio` Return the molecular ratio, $R_{\rm mol} = M_{\rm H_2}/M_{\rm HI}$. Must have the following interface:

```
double precision function myImplementationRatio(self,massISM,node)
 class           (outputAnalysisMolecularRatioClass), intent(inout) :: self
 double precision                                   , intent(in   ) :: massISM
 type            (treeNode                         ), intent(inout) :: node
end double precision function myImplementationRatio
```

`ratioScatter` Return the scatter in logarithmic molecular ratio, $\log_{10} R_{\rm mol} = \log_{10}(M_{\rm H_2}/M_{\rm HI})$. Must have the following interface:

```
double precision function myImplementationRatioScatter(self,massISM,node)
 class           (outputAnalysisMolecularRatioClass), intent(inout) :: self
 double precision                                   , intent(in   ) :: massISM
 type            (treeNode                         ), intent(inout) :: node
end double precision function myImplementationRatioScatter
```

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (outputAnalysisMolecularRatioClass), intent(inout)          :: self
  logical                                   , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (outputAnalysisMolecularRatioClass), intent(inout)          :: self
 type   (inputParameters                  ), intent(inout)          :: descriptor
 logical                                   , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(outputAnalysisMolecularRatioClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (outputAnalysisMolecularRatioClass), intent(inout) :: self
 integer                                   , intent(in   ) :: stateFile
 type   (fgsl_file                        ), intent(in   ) :: fgslStateFile
 integer(c_size_t                         ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (outputAnalysisMolecularRatioClass)                          , intent(inout)&
&   :: self
 type     (varying_string                   ), allocatable, dimension(:), intent(inout)&
&   :: allowedParameters
 character(len=*                            )                          , intent(in   )&
&   :: sourceName
 end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(outputAnalysisMolecularRatioClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(outputAnalysisMolecularRatioClass), intent(inout) :: self
 class(outputAnalysisMolecularRatioClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (outputAnalysisMolecularRatioClass), intent(inout) :: self
 integer                                   , intent(in   ) :: stateFile
 type   (fgsl_file                        ), intent(in   ) :: fgslStateFile
 integer(c_size_t                         ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`outputAnalysisMolecularRatioObreschkow2009` A high-pass filter analysis property operator class.

Output Analysis Property Operator

Additional implementations for output analysis property operator are added using the `outputAnalysisPropertyOperator` class. The implementation should be placed in a file containing the directive:

```
!# <outputAnalysisPropertyOperator name="outputAnalysisPropertyOperatorMyImplementation">
!# <description>A short description of the implementation.</description>
!# </outputAnalysisPropertyOperator>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `outputAnalysisPropertyOperatorMyImplementation`. The file *must* define a type that extends the `outputAnalysisPropertyOperatorClass` class (or extends another type which is itself an extension of the `outputAnalysisPropertyOperatorClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`operate` Operate on the given property. Must have the following interface:

```
 double precision function myImplementationOperate(self,propertyValue,node,propertyType,&
&  outputIndex)
  class           (outputAnalysisPropertyOperatorClass), intent(inout)           :: self
  double precision                                     , intent(in   )           :: &
&  propertyValue
  type            (treeNode                           ), intent(inout), optional :: node
  integer                                              , intent(inout), optional :: &
&  propertyType
  integer         (c_size_t                           ), intent(in   ), optional :: &
&  outputIndex
 end double precision function myImplementationOperate
```

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (outputAnalysisPropertyOperatorClass), intent(inout)           :: self
  logical                                     , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (outputAnalysisPropertyOperatorClass), intent(inout)           :: self
 type   (inputParameters                     ), intent(inout)           :: descriptor
 logical                                      , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(outputAnalysisPropertyOperatorClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (outputAnalysisPropertyOperatorClass), intent(inout) :: self
 integer                                     , intent(in   ) :: stateFile
 type   (fgsl_file                          ), intent(in   ) :: fgslStateFile
 integer(c_size_t                           ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (outputAnalysisPropertyOperatorClass)                          , intent(&
&  inout) :: self
 type     (varying_string                     ), allocatable, dimension(:), intent(&
&  inout) :: allowedParameters
 character(len=*                              )                          , intent(in  &
&   ) :: sourceName
 end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(outputAnalysisPropertyOperatorClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(outputAnalysisPropertyOperatorClass), intent(inout) :: self
 class(outputAnalysisPropertyOperatorClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (outputAnalysisPropertyOperatorClass), intent(inout) :: self
 integer                                     , intent(in   ) :: stateFile
 type   (fgsl_file                          ), intent(in   ) :: fgslStateFile
 integer(c_size_t                           ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`outputAnalysisPropertyOperatorAntiLog10` An anti-$\log_{10}()$ output analysis property operator class.

`outputAnalysisPropertyOperatorLog10` An log10 output analysis property operator class.

`outputAnalysisPropertyOperatorSystmtcPolynomial` A polynomial systematic shift output analysis property operator class.

`outputAnalysisPropertyOperatorCsmlgyAnglrDstnc` A cosmological angular distance corrector analysis property operator class.

`outputAnalysisPropertyOperatorMultiply` A high-pass filter analysis property operator class.

`outputAnalysisPropertyOperatorCsmlgyLmnstyDstnc` A cosmological luminosity distance corrector analysis property operator class.

`outputAnalysisPropertyOperatorMagnitude` An output analysis property operator class.

`outputAnalysisPropertyOperatorNormal` A property operator class in which the property value is replaced with an integral over a normal distribution between given limits, using the property value at the mean of the distribution.

`outputAnalysisPropertyOperatorBoolean` A boolean analysis property operator class, specifically $x \rightarrow x/|x|$, that is, the operator maintains the sign of the input while normalizing the magnitude to unity (or zero for zero input).

`outputAnalysisPropertyOperatorSequence` A sequence output analysis property operator class.

`outputAnalysisPropertyOperatorIdentity` An identity output analysis property operator class.

`outputAnalysisPropertyOperatorSquare` An square output analysis property operator class.

`outputAnalysisPropertyOperatorFilterHighPass` A high-pass filter analysis property operator class.

`outputAnalysisPropertyOperatorMetallicity12LogNH` A property operator class which converts a metallicity, assumed to be a mass ratio of a given element to hydrogen, to $12 + \log_{10}(\mathrm{N/H})$ form.

`outputAnalysisPropertyOperatorSquareRoot` An squareRoot output analysis property operator class.

`outputAnalysisPropertyOperatorHIMass` A conversion of ISM mass to HI mass analysis property operator class.

`outputAnalysisPropertyOperatorMinMax` A min-max analysis property operator class.

Output Analysis Weight Operator

Additional implementations for output analysis weight operator are added using the `outputAnalysisWeightOperator` class. The implementation should be placed in a file containing the directive:

```
!# <outputAnalysisWeightOperator name="outputAnalysisWeightOperatorMyImplementation">
!# <description>A short description of the implementation.</description>
!# </outputAnalysisWeightOperator>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `outputAnalysisWeightOperatorMyImplementation`. The file *must* define a type that extends the `outputAnalysisWeightOperatorClass` class (or extends another type which is itself an extension of the `outputAnalysisWeightOperatorClass`

class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`operate` Operate on the given weight. Must have the following interface:

```
 double precision function myImplementationOperate(self,weightValue,node,propertyValue, &
&  propertyValueIntrinsic,propertyType , propertyQuantity,outputIndex)
  class           (outputAnalysisWeightOperatorClass), intent(inout) :: self
  double precision                                   , intent(in   ) :: weightValue
  type            (treeNode                         ), intent(inout) :: node
  double precision                                   , intent(in   ) :: propertyValue, &
&  propertyValueIntrinsic
  integer                                            , intent(in   ) :: propertyType, &
&  propertyQuantity
  integer         (c_size_t                         ), intent(in   ) :: outputIndex
 end double precision function myImplementationOperate
```

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (outputAnalysisWeightOperatorClass), intent(inout)           :: self
  logical                                   , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (outputAnalysisWeightOperatorClass), intent(inout)           :: self
 type   (inputParameters                  ), intent(inout)           :: descriptor
 logical                                   , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(outputAnalysisWeightOperatorClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (outputAnalysisWeightOperatorClass), intent(inout) :: self
 integer                                   , intent(in   ) :: stateFile
 type   (fgsl_file                        ), intent(in   ) :: fgslStateFile
 integer(c_size_t                         ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (outputAnalysisWeightOperatorClass)                          , intent(inout)&
&   :: self
 type     (varying_string                  ), allocatable, dimension(:), intent(inout)&
&   :: allowedParameters
 character(len=*                           )                          , intent(in   )&
&   :: sourceName
end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(outputAnalysisWeightOperatorClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(outputAnalysisWeightOperatorClass), intent(inout) :: self
 class(outputAnalysisWeightOperatorClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (outputAnalysisWeightOperatorClass), intent(inout) :: self
 integer                                   , intent(in   ) :: stateFile
 type   (fgsl_file                         ), intent(in   ) :: fgslStateFile
 integer(c_size_t                          ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`outputAnalysisWeightOperatorFilterHighPass` A high-pass filter analysis weight operator class.

`outputAnalysisWeightOperatorProperty` An analysis weight operator class which weights by a property value.

`outputAnalysisWeightOperatorNormal` A weight operator class in which the weight is multiplied by an integral over a normal distribution.

`outputAnalysisWeightOperatorNbodyMass` A weight operator class in which the weight is multiplied by an integral over the N-body halo mass distribution.

`outputAnalysisWeightOperatorSequence` A sequence output analysis weight operator class.

`outputAnalysisWeightOperatorIdentity` An identity analysis weight operator class.

`outputAnalysisWeightOperatorCsmlgyVolume` A cosmological volume corrector analysis weight operator class.

Output Times

Additional implementations for output times are added using the `outputTimes` class. The implementation should be placed in a file containing the directive:

```
!# <outputTimes name="outputTimesMyImplementation">
!# <description>A short description of the implementation.</description>
!# </outputTimes>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `outputTimesMyImplementation`. The file *must* define a type that extends the `outputTimesClass` class (or extends another type which is itself an extension of the `outputTimesClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`timeNext` Given a `time`, return the time of the next output, and (optionally) the index of that output. Must have the following interface:

```
double precision function myImplementationTimeNext(self,timeCurrent,indexOutput)
 class           (outputTimesClass), intent(inout)           :: self
 double precision                  , intent(in   )           :: timeCurrent
 integer         (c_size_t        ), intent(  out), optional :: indexOutput
end double precision function myImplementationTimeNext
```

`count` Return the number of output times. Must have the following interface:

```
integer(c_size_t) function myImplementationCount(self)
 class(outputTimesClass), intent(inout) :: self
end integer(c_size_t) function myImplementationCount
```

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
 class  (outputTimesClass), intent(inout)           :: self
 logical                  , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (outputTimesClass), intent(inout)           :: self
 type   (inputParameters ), intent(inout)           :: descriptor
 logical                  , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(outputTimesClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`time` Return the output time index by `indexOutput`. Must have the following interface:

```
double precision function myImplementationTime(self,indexOutput)
 class  (outputTimesClass), intent(inout) :: self
 integer(c_size_t        ), intent(in   ) :: indexOutput
end double precision function myImplementationTime
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (outputTimesClass), intent(inout) :: self
 integer                  , intent(in   ) :: stateFile
 type   (fgsl_file       ), intent(in   ) :: fgslStateFile
 integer(c_size_t        ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (outputTimesClass)                          , intent(inout) :: self
 type     (varying_string  ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*           )                          , intent(in   ) :: sourceName
end subroutine myImplementationAllowedParameters
```

`index` Return the index of the output at the given `time`. If `findClosest` is given and is true then the closest matching output is returned. Must have the following interface:

```
integer(c_size_t) function myImplementationIndex(self,time,findClosest)
 class           (outputTimesClass), intent(inout)           :: self
 double precision                  , intent(in   )           :: time
 logical                           , intent(in   ), optional :: findClosest
end integer(c_size_t) function myImplementationIndex
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(outputTimesClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`redshift` Return the output redshift index by `indexOutput`. Must have the following interface:

```
double precision function myImplementationRedshift(self,indexOutput)
 class  (outputTimesClass), intent(inout) :: self
 integer(c_size_t        ), intent(in   ) :: indexOutput
end double precision function myImplementationRedshift
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(outputTimesClass), intent(inout) :: self
 class(outputTimesClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (outputTimesClass), intent(inout) :: self
 integer                  , intent(in   ) :: stateFile
 type   (fgsl_file       ), intent(in   ) :: fgslStateFile
 integer(c_size_t        ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

`timePrevious` Given a `time`, return the time of the previous output. Must have the following interface:

```
double precision function myImplementationTimePrevious(self,timeCurrent)
 class           (outputTimesClass), intent(inout) :: self
 double precision                  , intent(in   ) :: timeCurrent
end double precision function myImplementationTimePrevious
```

Existing implementations are:

`outputTimesList` An output times class which simply reads a list of output times from a parameter.

Posterior Sampling Convergence Criteria

Additional implementations for posterior sampling convergence criteria are added using the `posteriorSampleConvergence` class. The implementation should be placed in a file containing the directive:

```
!# <posteriorSampleConvergence name="posteriorSampleConvergenceMyImplementation">
!# <description>A short description of the implementation.</description>
!# </posteriorSampleConvergence>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `posteriorSampleConvergenceMyImplementation`. The file *must* define a type that extends the `posteriorSampleConvergenceClass` class (or extends another type which is itself an extension of the `posteriorSampleConvergenceClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`logReport` Log a report on convergence state to the given file unit. Must have the following interface:

```
subroutine myImplementationLogReport(self,fileUnit)
 class  (posteriorSampleConvergenceClass), intent(inout) :: self
 integer                                 , intent(in   ) :: fileUnit
end subroutine myImplementationLogReport
```

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (posteriorSampleConvergenceClass), intent(inout)          :: self
  logical                                 , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (posteriorSampleConvergenceClass), intent(inout)          :: self
 type   (inputParameters                ), intent(inout)          :: descriptor
 logical                                 , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(posteriorSampleConvergenceClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (posteriorSampleConvergenceClass), intent(inout) :: self
 integer                                 , intent(in   ) :: stateFile
 type   (fgsl_file                       ), intent(in   ) :: fgslStateFile
 integer(c_size_t                        ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (posteriorSampleConvergenceClass)                          , intent(inout) &
&  :: self
 type     (varying_string                 ), allocatable, dimension(:), intent(inout) &
&  :: allowedParameters
 character(len=*                          )                          , intent(in   ) &
&  :: sourceName
 end subroutine myImplementationAllowedParameters
```

`stateIsOutlier` Return true if the specified state is deemed to be an outlier. Must have the following interface:

```
logical function myImplementationStateIsOutlier(self,stateIndex)
 class  (posteriorSampleConvergenceClass), intent(inout) :: self
 integer                                 , intent(in   ) :: stateIndex
end logical function myImplementationStateIsOutlier
```

`reset` Reset the convergence calculation. Must have the following interface:

```
subroutine myImplementationReset(self)
 class(posteriorSampleConvergenceClass), intent(inout) :: self
end subroutine myImplementationReset
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(posteriorSampleConvergenceClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(posteriorSampleConvergenceClass), intent(inout) :: self
 class(posteriorSampleConvergenceClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`isConverged` Returns true if the posterior sampling is deemed to be converged. Must have the following interface:

```
logical function myImplementationIsConverged(self,simulationState,logLikelihood)
 class           (posteriorSampleConvergenceClass), intent(inout)           :: self
 class           (posteriorSampleStateClass      ), intent(inout), optional :: &
&  simulationState
 double precision                                 , intent(in   ), optional :: &
&  logLikelihood
 end logical function myImplementationIsConverged
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (posteriorSampleConvergenceClass), intent(inout) :: self
 integer                                 , intent(in   ) :: stateFile
 type   (fgsl_file                       ), intent(in   ) :: fgslStateFile
 integer(c_size_t                        ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

`convergedAtStep` Returns the step at which the sampling reached convergence. Must have the following interface:

```
integer function myImplementationConvergedAtStep(self)
 class(posteriorSampleConvergenceClass), intent(inout) :: self
end integer function myImplementationConvergedAtStep
```

Existing implementations are:

`posteriorSampleConvergenceGelmanRubin` A posterior sampling convergence class which implements the Gelman-Rubin statistic.

`posteriorSampleConvergenceNever` A posterior sampling convergence class which never converges.

`posteriorSampleConvergenceLikelihoodThreshold` A posterior sampling convergence class which declares convergence once all likelihoods are above a threshold.

Posterior Sampling Differential Evolution Proposal Size

Additional implementations for posterior sampling differential evolution proposal size are added using the `posteriorSampleDffrntlEvltnProposalSize` class. The implementation should be placed in a file containing the directive:

```
!# <posteriorSampleDffrntlEvltnProposalSize name="posteriorSampleDffrntlEvltnProposalSizeMyImplementatio
!# <description>A short description of the implementation.</description>
!# </posteriorSampleDffrntlEvltnProposalSize>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `posteriorSampleDffrntlEvltnProposalSizeMyImplementation`. The file *must* define a type that extends the `posteriorSampleDffrntlEvltnProposalSizeClass` class (or extends another type which is itself an extension of the `posteriorSampleDffrntlEvltnProposalSizeClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (posteriorSampleDffrntlEvltnProposalSizeClass), intent(inout)          :: self
  logical                                              , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationDescriptor(self,descriptor,includeMethod)
  class  (posteriorSampleDffrntlEvltnProposalSizeClass), intent(inout)          :: self
  type   (inputParameters                              ), intent(inout)          :: &
&  descriptor
  logical                                              , intent(in   ), optional :: &
&  includeMethod
 end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAutoHook(self)
  class(posteriorSampleDffrntlEvltnProposalSizeClass), intent(inout) :: self
 end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
  class  (posteriorSampleDffrntlEvltnProposalSizeClass), intent(inout) :: self
```

```
 integer                                                , intent(in   ) :: stateFile
 type   (fgsl_file                                     ), intent(in   ) :: fgslStateFile
 integer(c_size_t                                      ), intent(in   ) :: &
&  stateOperationID
 end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
  class    (posteriorSampleDffrntlEvltnProposalSizeClass)                         , &
&  intent(inout) :: self
  type     (varying_string                               ), allocatable, dimension(:), &
&  intent(inout) :: allowedParameters
  character(len=*                                        )                         , &
&  intent(in   ) :: sourceName
 end subroutine myImplementationAllowedParameters
```

`gamma` Sample from the jump distribution. Must have the following interface:

```
 double precision function myImplementationGamma(self,simulationState,&
&  simulationConvergence)
  class(posteriorSampleDffrntlEvltnProposalSizeClass), intent(inout) :: self
  class(posteriorSampleStateClass                   ), intent(inout) :: simulationState
  class(posteriorSampleConvergenceClass             ), intent(inout) :: &
&  simulationConvergence
 end double precision function myImplementationGamma
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationObjectType(self)
  class(posteriorSampleDffrntlEvltnProposalSizeClass), intent(inout) :: self
 end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationDeepCopy(self,destination)
  class(posteriorSampleDffrntlEvltnProposalSizeClass), intent(inout) :: self
  class(posteriorSampleDffrntlEvltnProposalSizeClass), intent(inout) :: destination
 end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
  class  (posteriorSampleDffrntlEvltnProposalSizeClass), intent(inout) :: self
  integer                                               , intent(in   ) :: stateFile
  type   (fgsl_file                                     ), intent(in   ) :: fgslStateFile
  integer(c_size_t                                      ), intent(in   ) :: &
&  stateOperationID
 end subroutine myImplementationStateRestore
```

Existing implementations are:

`posteriorSampleDffrntlEvltnProposalSizeFixed` A posterior sampling differential evolution proposal size class in which the proposal size is fixed.

`posteriorSampleDffrntlEvltnProposalSizeAdaptive` A posterior sampling differential evolution proposal size class in which the proposal size is adaptive.

Posterior Sampling Differential Evolution Proposal Size Temperature Exponent

Additional implementations for posterior sampling differential evolution proposal size temperature exponent are added using the `posteriorSampleDffrntlEvltnPrpslSzTmpExp` class. The implementation should be placed in a file containing the directive:

```
!# <posteriorSampleDffrntlEvltnPrpslSzTmpExp name="posteriorSampleDffrntlEvltnPrpslSzTmpExpMyImplementat
!# <description>A short description of the implementation.</description>
!# </posteriorSampleDffrntlEvltnPrpslSzTmpExp>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `posteriorSampleDffrntlEvltnPrpslSzTmpExpMyImplementation`. The file *must* define a type that extends the `posteriorSampleDffrntlEvltnPrpslSzTmpExpClass` class (or extends another type which is itself an extension of the `posteriorSampleDffrntlEvltnPrpslSzTmpExpClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&)
  class  (posteriorSampleDffrntlEvltnPrpslSzTmpExpClass), intent(inout)           :: &
&  self
  logical                                               , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationDescriptor(self,descriptor,includeMethod)
  class  (posteriorSampleDffrntlEvltnPrpslSzTmpExpClass), intent(inout)           :: &
&  self
  type   (inputParameters                               ), intent(inout)           :: &
&  descriptor
  logical                                               , intent(in   ), optional :: &
&  includeMethod
 end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(posteriorSampleDffrntlEvltnPrpslSzTmpExpClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (posteriorSampleDffrntlEvltnPrpslSzTmpExpClass), intent(inout) :: self
 integer                                               , intent(in   ) :: stateFile
 type   (fgsl_file                                     ), intent(in   ) :: fgslStateFile
 integer(c_size_t                                      ), intent(in   ) :: &
&  stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (posteriorSampleDffrntlEvltnPrpslSzTmpExpClass)                          , &
&  intent(inout) :: self
 type     (varying_string                                ), allocatable, dimension(:), &
&  intent(inout) :: allowedParameters
 character(len=*                                         )                          , &
&  intent(in   ) :: sourceName
end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(posteriorSampleDffrntlEvltnPrpslSzTmpExpClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(posteriorSampleDffrntlEvltnPrpslSzTmpExpClass), intent(inout) :: self
 class(posteriorSampleDffrntlEvltnPrpslSzTmpExpClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`exponent` Return the temperature exponent. Must have the following interface:

```
double precision function myImplementationExponent(self,temperedStates,temperatures,&
&  simulationState,simulationConvergence)
 class           (posteriorSampleDffrntlEvltnPrpslSzTmpExpClass)              , intent(&
&  inout) :: self
 class           (posteriorSampleStateClass                   ), dimension(:), intent(&
&  inout) :: temperedStates
 double precision                                               , dimension(:), intent(&
&  in   ) :: temperatures
 class           (posteriorSampleStateClass                   )              , intent(&
&  inout) :: simulationState
```

```
     class           (posteriorSampleConvergenceClass             )              , intent(&
    &  inout) :: simulationConvergence
     end double precision function myImplementationExponent
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
    subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
     class  (posteriorSampleDffrntlEvltnPrpslSzTmpExpClass), intent(inout) :: self
     integer                                               , intent(in   ) :: stateFile
     type   (fgsl_file                                     ), intent(in   ) :: fgslStateFile
     integer(c_size_t                                      ), intent(in   ) :: &
    &  stateOperationID
     end subroutine myImplementationStateRestore
```

Existing implementations are:

`posteriorSampleDffrntlEvltnPrpslSzTmpExpAdaptive` A posterior sampling differential evolution proposal size class in which the exponent is adaptive.

`posteriorSampleDffrntlEvltnPrpslSzTmpExpFixed` A posterior sampling differential evolution proposal size class in which the exponent is fixed.

Posterior Sampling Differential Evolution Random Jumps

Additional implementations for posterior sampling differential evolution random jumps are added using the `posteriorSampleDffrntlEvltnRandomJump` class. The implementation should be placed in a file containing the directive:

```
!# <posteriorSampleDffrntlEvltnRandomJump name="posteriorSampleDffrntlEvltnRandomJumpMyImplementation">
!# <description>A short description of the implementation.</description>
!# </posteriorSampleDffrntlEvltnRandomJump>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `posteriorSampleDffrntlEvltnRandomJumpMyImplementation`. The file *must* define a type that extends the `posteriorSampleDffrntlEvltnRandomJumpClass` class (or extends another type which is itself an extension of the `posteriorSampleDffrntlEvltnRandomJumpClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
    type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
    &  )
     class  (posteriorSampleDffrntlEvltnRandomJumpClass), intent(inout)          :: self
     logical                                            , intent(in   ), optional :: &
    &  includeSourceDigest
     end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (posteriorSampleDffrntlEvltnRandomJumpClass), intent(inout)           :: self
 type   (inputParameters                            ), intent(inout)           :: &
&  descriptor
 logical                                             , intent(in   ), optional :: &
&  includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(posteriorSampleDffrntlEvltnRandomJumpClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (posteriorSampleDffrntlEvltnRandomJumpClass), intent(inout) :: self
 integer                                             , intent(in   ) :: stateFile
 type   (fgsl_file                                   ), intent(in   ) :: fgslStateFile
 integer(c_size_t                                    ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (posteriorSampleDffrntlEvltnRandomJumpClass)                          , &
&  intent(inout) :: self
 type     (varying_string                            ), allocatable, dimension(:), &
&  intent(inout) :: allowedParameters
 character(len=*                                     )                          , &
&  intent(in   ) :: sourceName
end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(posteriorSampleDffrntlEvltnRandomJumpClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(posteriorSampleDffrntlEvltnRandomJumpClass), intent(inout) :: self
 class(posteriorSampleDffrntlEvltnRandomJumpClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`sample` Sample from the jump distribution. Must have the following interface:

```
 double precision, dimension(size(modelParameters_)) function myImplementationSample(&
&  self,modelParameters_,simulationState)
  class(posteriorSampleDffrntlEvltnRandomJumpClass)              , intent(inout) :: self
  type (modelParameterList                        ), dimension(:), intent(in   ) :: &
&  modelParameters_
  class(posteriorSampleStateClass                 )              , intent(inout) :: &
&  simulationState
 end double precision, dimension(size(modelParameters_)) function myImplementationSample
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
  class  (posteriorSampleDffrntlEvltnRandomJumpClass), intent(inout) :: self
  integer                                            , intent(in   ) :: stateFile
  type   (fgsl_file                                  ), intent(in   ) :: fgslStateFile
  integer(c_size_t                                   ), intent(in   ) :: stateOperationID
 end subroutine myImplementationStateRestore
```

Existing implementations are:

`posteriorSampleDffrntlEvltnRandomJumpSimple` A posterior sampling differential evolution random jump class in which the jump is drawn from a fixed distribution.

`posteriorSampleDffrntlEvltnRandomJumpAdaptive` A posterior sampling differential evolution random jump class in which the jump is drawn from an adaptive distribution which scales with the range spanned by the sample states..

Posterior Sampling Likelihoods

Additional implementations for posterior sampling likelihoods are added using the `posteriorSampleLikelihood` class. The implementation should be placed in a file containing the directive:

```
!# <posteriorSampleLikelihood name="posteriorSampleLikelihoodMyImplementation">
!# <description>A short description of the implementation.</description>
!# </posteriorSampleLikelihood>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `posteriorSampleLikelihoodMyImplementation`. The file *must* define a type that extends the `posteriorSampleLikelihoodClass` class (or extends another type which is itself an extension of the `posteriorSampleLikelihoodClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (posteriorSampleLikelihoodClass), intent(inout)          :: self
  logical                                , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (posteriorSampleLikelihoodClass), intent(inout)          :: self
 type   (inputParameters               ), intent(inout)          :: descriptor
 logical                                , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(posteriorSampleLikelihoodClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (posteriorSampleLikelihoodClass), intent(inout) :: self
 integer                                , intent(in   ) :: stateFile
 type   (fgsl_file                      ), intent(in   ) :: fgslStateFile
 integer(c_size_t                       ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`evaluate` Evaluate the likelihood. Must have the following interface:

```
 double precision function myImplementationEvaluate(self,simulationState,&
&  modelParametersActive_, modelParametersInactive_,simulationConvergence,temperature, &
&  logLikelihoodCurrent, logPriorCurrent, logPriorProposed,timeEvaluate,&
&  logLikelihoodVariance,forceAcceptance)
  class           (posteriorSampleLikelihoodClass )              , intent(inout)        &
&     :: self
  class           (posteriorSampleStateClass      )              , intent(inout)        &
&     :: simulationState
  type            (modelParameterList             ), dimension(:), intent(in   )        &
&     :: modelParametersActive_, modelParametersInactive_
  class           (posteriorSampleConvergenceClass)              , intent(inout)        &
&     :: simulationConvergence
  double precision                                               , intent(in   )        &
&     :: temperature, logLikelihoodCurrent, logPriorCurrent, logPriorProposed
  real                                                           , intent(inout)        &
&     :: timeEvaluate
  double precision                                               , intent(  out), &
&  optional :: logLikelihoodVariance
  logical                                                        , intent(inout), &
&  optional :: forceAcceptance
 end double precision function myImplementationEvaluate
```

`restore` Log a report on convergence state to the given file unit. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationRestore(self,simulationState,logLikelihood)
 class           (posteriorSampleLikelihoodClass)              , intent(inout) :: self
 double precision                                 , dimension(:), intent(in   ) :: &
&  simulationState
 double precision                                               , intent(in   ) :: &
&  logLikelihood
end subroutine myImplementationRestore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (posteriorSampleLikelihoodClass)                          , intent(inout) ::&
&   self
 type     (varying_string               ), allocatable, dimension(:), intent(inout) ::&
&   allowedParameters
 character(len=*                        )                          , intent(in   ) ::&
&   sourceName
end subroutine myImplementationAllowedParameters
```

`functionChanged` Respond to possible changes in the likelihood function. Must have the following interface:

```
subroutine myImplementationFunctionChanged(self)
 class(posteriorSampleLikelihoodClass), intent(inout) :: self
end subroutine myImplementationFunctionChanged
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(posteriorSampleLikelihoodClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`willEvaluate` Returns true if the likelihood will be evaluated for this state. A default implementation exists. If overridden the following interface must be used:

```
logical function myImplementationWillEvaluate(self,simulationState,modelParameters_,&
&  simulationConvergence,temperature, logLikelihoodCurrent, logPriorCurrent, &
&  logPriorProposed)
 class           (posteriorSampleLikelihoodClass )              , intent(inout) :: self
 class           (posteriorSampleStateClass      )              , intent(inout) :: &
&  simulationState
 type            (modelParameterList             ), dimension(:), intent(in   ) :: &
&  modelParameters_
 class           (posteriorSampleConvergenceClass)              , intent(inout) :: &
&  simulationConvergence
 double precision                                               , intent(in   ) :: &
&  temperature, logLikelihoodCurrent, logPriorCurrent, logPriorProposed
end logical function myImplementationWillEvaluate
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(posteriorSampleLikelihoodClass), intent(inout) :: self
 class(posteriorSampleLikelihoodClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (posteriorSampleLikelihoodClass), intent(inout) :: self
 integer                                , intent(in   ) :: stateFile
 type   (fgsl_file                      ), intent(in   ) :: fgslStateFile
 integer(c_size_t                       ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`posteriorSampleLikelihoodMassFunction` A posterior sampling likelihood class which implements a likelihood for mass functions.

`posteriorSampleLikelihoodIndpndntLklhdsSqntl` A posterior sampling likelihood class which sequentially combines other likelihoods assumed to be independent. This class begins by evaluating the first likelihood. If the likelihood is negative, then it is immediately returned, without evaluation of any further likelihoods. If it is positive, then the next likelihood is evaluated and the same conditions applied. This process repeats until either a negative likelihood is found, or all likelihoods are evaluated. Once a given likelihood has been evaluated it will be evaluated on all subsequent calls. Additionally, when a new likelihood is evaluated for the first time, acceptance of the proposed state will be forced. This class therefore allows a sequence of likelihoods to be specified which must be sequentially made sufficiently "good" before evaluating the next. The approach is intended to allow crude, but rapid constraints to be placed on parameters before progressing to more detailed, but slow to evaluate constraints.

`posteriorSampleLikelihoodHaloMassFunction` A posterior sampling likelihood class which implements a likelihood for halo mass functions.

`posteriorSampleLikelihoodPrjctdCorrelationFunction` A posterior sampling likelihood class which implements a likelihood for projected correlation functions.

`posteriorSampleLikelihoodGaussianRegression` A posterior sampling likelihood class which implements a likelihood using Gaussian regression to emulate another likelihood.

`posteriorSampleLikelihoodMltiVrtNormalStochastic` A posterior sampling likelihood class which implements a multivariate normal.

`posteriorSampleLikelihoodPosteriorAsPrior` A posterior sampling likelihood class which implements a likelihood using a given posterior distribution over the parameters in the form of a set of MCMC chains.

`posteriorSampleLikelihoodSEDFit` A posterior sampling likelihood class which implements a likelihood for SED fitting.

`posteriorSampleLikelihoodIndependentLikelihoods` A posterior sampling likelihood class which combines other likelihoods assumed to be independent.

`posteriorSampleLikelihoodSpinDistribution` A posterior sampling likelihood class which implements a likelihood for halo spin distributions.

`posteriorSampleLikelihoodGalaxyPopulation` A posterior sampling likelihood class which implements a likelihood for Galacticus models.

`posteriorSampleLikelihoodMultivariateNormal` A posterior sampling likelihood class which implements a multivariate normal.

Posterior Sampling Simulations

Additional implementations for posterior sampling simulations are added using the `posteriorSampleSimulation` class. The implementation should be placed in a file containing the directive:

```
!# <posteriorSampleSimulation name="posteriorSampleSimulationMyImplementation">
!# <description>A short description of the implementation.</description>
!# </posteriorSampleSimulation>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `posteriorSampleSimulationMyImplementation`. The file *must* define a type that extends the `posteriorSampleSimulationClass` class (or extends another type which is itself an extension of the `posteriorSampleSimulationClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (posteriorSampleSimulationClass), intent(inout)          :: self
  logical                                , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (posteriorSampleSimulationClass), intent(inout)          :: self
 type   (inputParameters               ), intent(inout)          :: descriptor
 logical                                , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(posteriorSampleSimulationClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (posteriorSampleSimulationClass), intent(inout) :: self
 integer                                , intent(in   ) :: stateFile
 type   (fgsl_file                      ), intent(in   ) :: fgslStateFile
 integer(c_size_t                       ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (posteriorSampleSimulationClass)                          , intent(inout) ::&
&  self
 type     (varying_string               ), allocatable, dimension(:), intent(inout) ::&
&  allowedParameters
 character(len=*                         )                          , intent(in   ) ::&
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`simulate` Perform the simulation. Must have the following interface:

```
subroutine myImplementationSimulate(self)
 class(posteriorSampleSimulationClass), intent(inout) :: self
end subroutine myImplementationSimulate
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(posteriorSampleSimulationClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(posteriorSampleSimulationClass), intent(inout) :: self
 class(posteriorSampleSimulationClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (posteriorSampleSimulationClass), intent(inout) :: self
 integer                                , intent(in   ) :: stateFile
 type   (fgsl_file                      ), intent(in   ) :: fgslStateFile
 integer(c_size_t                       ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`posteriorSampleSimulationAnnealedDffrntlEvltn` A posterior sampling simulation class which implements an annealed differential evolution algorithm.

`posteriorSampleSimulationParticleSwarm` A posterior sampling simulation class which implements the particle swarm algorithm.

`posteriorSampleSimulationStochasticDffrntlEvltn` A posterior sampling simulation class which implements a stochastic differential evolution algorithm.

`posteriorSampleSimulationDifferentialEvolution` A posterior sampling simulation class which implements the differential evolution algorithm.

`posteriorSampleSimulationTemperedDffrntlEvltn` A posterior sampling simulation class which implements a tempered differential evolution algorithm.

Posterior Sampling State

Additional implementations for posterior sampling state are added using the `posteriorSampleState` class. The implementation should be placed in a file containing the directive:

```
!# <posteriorSampleState name="posteriorSampleStateMyImplementation">
!# <description>A short description of the implementation.</description>
!# </posteriorSampleState>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `posteriorSampleStateMyImplementation`. The file *must* define a type that extends the `posteriorSampleStateClass` class (or extends another type which is itself an extension of the `posteriorSampleStateClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`chainIndexSet` Set the index of the chain associated with this state. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationChainIndexSet(self,chainIndex)
 class  (posteriorSampleStateClass), intent(inout) :: self
 integer                           , intent(in   ) :: chainIndex
end subroutine myImplementationChainIndexSet
```

`restore` Restore the state, one step at a time. Must have the following interface:

```
subroutine myImplementationRestore(self,stateVector,first)
 class           (posteriorSampleStateClass)              , intent(inout) :: self
 double precision                           , dimension(:), intent(in   ) :: &
&  stateVector
 logical                                                  , intent(in   ) :: first
end subroutine myImplementationRestore
```

`chainIndex` Return the index of the chain associated with this state. A default implementation exists. If overridden the following interface must be used:

```
integer function myImplementationChainIndex(self)
 class(posteriorSampleStateClass), intent(inout) :: self
end integer function myImplementationChainIndex
```

`acceptanceRate` Return the state acceptance rate. Must have the following interface:

```
double precision function myImplementationAcceptanceRate(self)
 class(posteriorSampleStateClass), intent(inout) :: self
end double precision function myImplementationAcceptanceRate
```

`reset` Reset the state object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationReset(self)
 class(posteriorSampleStateClass), intent(inout) :: self
end subroutine myImplementationReset
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(posteriorSampleStateClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`dimension` Returns the dimension of the state. A default implementation exists. If overridden the following interface must be used:

```
integer function myImplementationDimension(self)
 class(posteriorSampleStateClass), intent(inout) :: self
end integer function myImplementationDimension
```

`get` Get the current state vector. Must have the following interface:

```
double precision, dimension(self%parameterCount) function myImplementationGet(self)
 class(posteriorSampleStateClass), intent(inout) :: self
end double precision, dimension(self%parameterCount) function myImplementationGet
```

`count` Returns the number of steps in the current state. A default implementation exists. If overridden the following interface must be used:

```
integer function myImplementationCount(self)
 class(posteriorSampleStateClass), intent(inout) :: self
end integer function myImplementationCount
```

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
 class  (posteriorSampleStateClass), intent(inout)          :: self
 logical                           , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`variance` Return the variance in the state vector. Must have the following interface:

```
double precision, dimension(self%parameterCount) function myImplementationVariance(self&
&  )
 class(posteriorSampleStateClass), intent(inout) :: self
end double precision, dimension(self%parameterCount) function myImplementationVariance
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (posteriorSampleStateClass), intent(inout)           :: self
 type   (inputParameters          ), intent(inout)           :: descriptor
 logical                           , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(posteriorSampleStateClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`parameterCountSet` Set the number of parameters in this state. Must have the following interface:

```
subroutine myImplementationParameterCountSet(self,parameterCount)
 class  (posteriorSampleStateClass), intent(inout) :: self
 integer                           , intent(in   ) :: parameterCount
end subroutine myImplementationParameterCountSet
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (posteriorSampleStateClass), intent(inout) :: self
 integer                           , intent(in   ) :: stateFile
 type   (fgsl_file                 ), intent(in   ) :: fgslStateFile
 integer(c_size_t                  ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (posteriorSampleStateClass)                          , intent(inout) :: self
 type     (varying_string           ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                    )                          , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`mean` Return the mean state. Must have the following interface:

```
double precision, dimension(self%parameterCount) function myImplementationMean(self)
 class(posteriorSampleStateClass), intent(inout) :: self
end double precision, dimension(self%parameterCount) function myImplementationMean
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(posteriorSampleStateClass), intent(inout) :: self
 class(posteriorSampleStateClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (posteriorSampleStateClass), intent(inout) :: self
 integer                           , intent(in   ) :: stateFile
 type   (fgsl_file                ), intent(in   ) :: fgslStateFile
 integer(c_size_t                 ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

`update` Update the state vector. Must have the following interface:

```
subroutine myImplementationUpdate(self,stateNew,logState,isConverged,outlierMask)
 class           (posteriorSampleStateClass)              , intent(inout)           :: &
&  self
 double precision                                 , dimension(:), intent(in   )           :: &
&  stateNew
 logical                                                        , intent(in   )           :: &
&  logState
 logical                                                        , intent(in   )           :: &
&  isConverged
 logical                                          , dimension(:), intent(in   ), optional :: &
&  outlierMask
 end subroutine myImplementationUpdate
```

Existing implementations are:

`posteriorSampleStateHistory` A posterior sampling state class which stores history.

`posteriorSampleStateSimple` A simple posterior sampling state class.

`posteriorSampleStateCorrelation` A correlation posterior sampling state class.

Posterior Sampling State Initialization

Additional implementations for posterior sampling state initialization are added using the `posteriorSampleStateInitialize` class. The implementation should be placed in a file containing the directive:

```
!# <posteriorSampleStateInitialize name="posteriorSampleStateInitializeMyImplementation">
!# <description>A short description of the implementation.</description>
!# </posteriorSampleStateInitialize>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `posteriorSampleStateInitializeMyImplementation`. The file *must* define a type that extends the

posteriorSampleStateInitializeClass class (or extends another type which is itself an extension of the posteriorSampleStateInitializeClass class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

hashedDescriptor Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (posteriorSampleStateInitializeClass), intent(inout)           :: self
  logical                                     , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

descriptor Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationDescriptor(self,descriptor,includeMethod)
  class  (posteriorSampleStateInitializeClass), intent(inout)           :: self
  type   (inputParameters                    ), intent(inout)           :: descriptor
  logical                                     , intent(in   ), optional :: includeMethod
 end subroutine myImplementationDescriptor
```

autoHook Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAutoHook(self)
  class(posteriorSampleStateInitializeClass), intent(inout) :: self
 end subroutine myImplementationAutoHook
```

stateStore Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
  class  (posteriorSampleStateInitializeClass), intent(inout) :: self
  integer                                     , intent(in   ) :: stateFile
  type   (fgsl_file                          ), intent(in   ) :: fgslStateFile
  integer(c_size_t                           ), intent(in   ) :: stateOperationID
 end subroutine myImplementationStateStore
```

initialize Initialize the state of the posterior sampler. Must have the following interface:

```
 subroutine myImplementationInitialize(self,simulationState,modelParameters_,&
&  modelLikelihood,timeEvaluatePrevious, logLikelihood, logPosterior)
  class           (posteriorSampleStateInitializeClass)              , intent(inout) :: &
&  self
  class           (posteriorSampleStateClass          )              , intent(inout) :: &
&  simulationState
  type            (modelParameterList                 ), dimension(:), intent(in   ) :: &
&  modelParameters_
  class           (posteriorSampleLikelihoodClass     )              , intent(inout) :: &
&  modelLikelihood
```

```
  double precision                                                    , intent(  out) :: &
&  timeEvaluatePrevious, logLikelihood, logPosterior
 end subroutine myImplementationInitialize
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
  class    (posteriorSampleStateInitializeClass)                          , intent(&
&  inout) :: self
  type     (varying_string                    ), allocatable, dimension(:), intent(&
&  inout) :: allowedParameters
  character(len=*                             )                          , intent(in  &
&   ) :: sourceName
 end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationObjectType(self)
  class(posteriorSampleStateInitializeClass), intent(inout) :: self
 end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationDeepCopy(self,destination)
  class(posteriorSampleStateInitializeClass), intent(inout) :: self
  class(posteriorSampleStateInitializeClass), intent(inout) :: destination
 end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
  class  (posteriorSampleStateInitializeClass), intent(inout) :: self
  integer                                     , intent(in   ) :: stateFile
  type   (fgsl_file                          ), intent(in   ) :: fgslStateFile
  integer(c_size_t                           ), intent(in   ) :: stateOperationID
 end subroutine myImplementationStateRestore
```

Existing implementations are:

`posteriorSampleStateInitializePriorRandom` A posterior sampling state initialization class which samples the inital state at random from the priors.

`posteriorSampleStateInitializeSwitched` A posterior sampling state initialization class which sets initial state by switching between two other options.

`posteriorSampleStateInitializeResume` A posterior sampling state initialization class which sets initial state to that at the end of a previous simulation.

`posteriorSampleStateInitializeLatinHypercube` A posterior sampling state initialization class which samples the inital state at random from the priors using Latin Hypercube sampling.

Posterior Sampling StoppingCriterion Criteria

Additional implementations for posterior sampling stoppingcriterion criteria are added using the `posteriorSampleStoppingCr` class. The implementation should be placed in a file containing the directive:

```
!# <posteriorSampleStoppingCriterion name="posteriorSampleStoppingCriterionMyImplementation">
!# <description>A short description of the implementation.</description>
!# </posteriorSampleStoppingCriterion>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `posteriorSampleStoppingCriterionMyImplementation`. The file *must* define a type that extends the `posteriorSampleStoppingCriterionClass` class (or extends another type which is itself an extension of the `posteriorSampleStoppingCriterionClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (posteriorSampleStoppingCriterionClass), intent(inout)           :: self
  logical                                       , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationDescriptor(self,descriptor,includeMethod)
  class  (posteriorSampleStoppingCriterionClass), intent(inout)           :: self
  type   (inputParameters                      ), intent(inout)           :: descriptor
  logical                                       , intent(in   ), optional :: &
&  includeMethod
 end subroutine myImplementationDescriptor
```

`stop` Returns true if the posterior sampling should stop. Must have the following interface:

```
 logical function myImplementationStop(self,simulationState)
  class(posteriorSampleStoppingCriterionClass), intent(inout) :: self
  class(posteriorSampleStateClass            ), intent(inout) :: simulationState
 end logical function myImplementationStop
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAutoHook(self)
  class(posteriorSampleStoppingCriterionClass), intent(inout) :: self
 end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (posteriorSampleStoppingCriterionClass), intent(inout) :: self
 integer                                       , intent(in   ) :: stateFile
 type   (fgsl_file                            ), intent(in   ) :: fgslStateFile
 integer(c_size_t                             ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (posteriorSampleStoppingCriterionClass)                          , intent(&
&  inout) :: self
 type     (varying_string                       ), allocatable, dimension(:), intent(&
&  inout) :: allowedParameters
 character(len=*                                 )                          , intent(in&
&     ) :: sourceName
 end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(posteriorSampleStoppingCriterionClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(posteriorSampleStoppingCriterionClass), intent(inout) :: self
 class(posteriorSampleStoppingCriterionClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (posteriorSampleStoppingCriterionClass), intent(inout) :: self
 integer                                       , intent(in   ) :: stateFile
 type   (fgsl_file                            ), intent(in   ) :: fgslStateFile
 integer(c_size_t                             ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`posteriorSampleStoppingCriterionStepCount` A posterior sampling stopping class which stepCount converges.

`posteriorSampleStoppingCriterionNever` A posterior sampling stopping class which never converges.

`posteriorSampleStoppingCriterionCorrelationLength` A posterior sampling stopping class which stops after a given number of correlation lengths.

Linear Theory Power Spectrum

Additional implementations for linear theory power spectrum are added using the `powerSpectrum` class. The implementation should be placed in a file containing the directive:

```
!# <powerSpectrum name="powerSpectrumMyImplementation">
!# <description>A short description of the implementation.</description>
!# </powerSpectrum>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `powerSpectrumMyImplementation`. The file *must* define a type that extends the `powerSpectrumClass` class (or extends another type which is itself an extension of the `powerSpectrumClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
 class  (powerSpectrumClass), intent(inout)           :: self
 logical                    , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (powerSpectrumClass), intent(inout)           :: self
 type   (inputParameters   ), intent(inout)           :: descriptor
 logical                    , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(powerSpectrumClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (powerSpectrumClass), intent(inout) :: self
 integer                    , intent(in   ) :: stateFile
 type   (fgsl_file         ), intent(in   ) :: fgslStateFile
 integer(c_size_t          ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (powerSpectrumClass)                          , intent(inout) :: self
 type     (varying_string    ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*             )                          , intent(in   ) :: sourceName
end subroutine myImplementationAllowedParameters
```

`powerLogarithmicDerivative` Return the logarithmic derivative of the power spectrum, $\mathrm{d}\ln P(k)/\mathrm{d}\ln k$, for $k =$`wavenumber` [Mpc^{-1}]. Must have the following interface:

```
double precision function myImplementationPowerLogarithmicDerivative(self,wavenumber)
 class           (powerSpectrumClass), intent(inout) :: self
 double precision                    , intent(in   ) :: wavenumber
end double precision function myImplementationPowerLogarithmicDerivative
```

`powerDimensionless` Return the dimensionless linear power spectrum for $k =$`wavenumber` [Mpc^{-1}]. Must have the following interface:

```
double precision function myImplementationPowerDimensionless(self,wavenumber)
 class           (powerSpectrumClass), intent(inout) :: self
 double precision                    , intent(in   ) :: wavenumber
end double precision function myImplementationPowerDimensionless
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(powerSpectrumClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(powerSpectrumClass), intent(inout) :: self
 class(powerSpectrumClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`power` Return the linear power spectrum for $k =$`wavenumber` [Mpc^{-1}]. Must have the following interface:

```
double precision function myImplementationPower(self,wavenumber)
 class           (powerSpectrumClass), intent(inout) :: self
 double precision                    , intent(in   ) :: wavenumber
end double precision function myImplementationPower
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (powerSpectrumClass), intent(inout) :: self
 integer                    , intent(in   ) :: stateFile
 type   (fgsl_file         ), intent(in   ) :: fgslStateFile
 integer(c_size_t          ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`powerSpectrumStandard` Provides a linear theory power spectrum class in which the power spectrum is just the transferred primordial power spectrum correctly normalized to $z = 0$.

Nonlinear Power Spectrum

Additional implementations for nonlinear power spectrum are added using the `powerSpectrumNonlinear` class. The implementation should be placed in a file containing the directive:

```
!# <powerSpectrumNonlinear name="powerSpectrumNonlinearMyImplementation">
!# <description>A short description of the implementation.</description>
!# </powerSpectrumNonlinear>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `powerSpectrumNonlinearMyImplementation`. The file *must* define a type that extends the `powerSpectrumNonlinearClass` class (or extends another type which is itself an extension of the `powerSpectrumNonlinearClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
 class  (powerSpectrumNonlinearClass), intent(inout)           :: self
 logical                             , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (powerSpectrumNonlinearClass), intent(inout)           :: self
 type   (inputParameters            ), intent(inout)           :: descriptor
 logical                             , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(powerSpectrumNonlinearClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`value` Return the nonlinear power spectrum for $k =$`wavenumber` [Mpc^{-1}] at cosmic time $t =$`time` [Gyr]. Must have the following interface:

```
double precision function myImplementationValue(self,wavenumber, time)
 class           (powerSpectrumNonlinearClass), intent(inout) :: self
 double precision                             , intent(in   ) :: wavenumber, time
end double precision function myImplementationValue
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (powerSpectrumNonlinearClass), intent(inout) :: self
 integer                             , intent(in   ) :: stateFile
 type   (fgsl_file                  ), intent(in   ) :: fgslStateFile
 integer(c_size_t                   ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (powerSpectrumNonlinearClass)                          , intent(inout) :: &
&  self
 type     (varying_string             ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                      )                          , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(powerSpectrumNonlinearClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(powerSpectrumNonlinearClass), intent(inout) :: self
 class(powerSpectrumNonlinearClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (powerSpectrumNonlinearClass), intent(inout) :: self
 integer                             , intent(in   ) :: stateFile
 type   (fgsl_file                  ), intent(in   ) :: fgslStateFile
 integer(c_size_t                   ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`powerSpectrumNonlinearPeacockDodds1996` Provides a nonlinear power spectrum class in which the power spectrum is computed using the algorithm of Peacock and Dodds [1996]. See §12.6.4.

`powerSpectrumNonlinearLinear` Provides a nonlinear power spectrum class in which the power spectrum equals the linear theory power spectrum. Intended primarily for testing purposes. See §12.6.4.

`powerSpectrumNonlinearCosmicEmu` Provides a nonlinear power spectrum class in which the power spectrum is computed using the code of Lawrence et al. [2010]. See §12.6.4.

Primordial Power Spectrum

Additional implementations for primordial power spectrum are added using the `powerSpectrumPrimordial` class. The implementation should be placed in a file containing the directive:

```
!# <powerSpectrumPrimordial name="powerSpectrumPrimordialMyImplementation">
!# <description>A short description of the implementation.</description>
!# </powerSpectrumPrimordial>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `powerSpectrumPrimordialMyImplementation`. The file *must* define a type that extends the `powerSpectrumPrimordialClass` class (or extends another type which is itself an extension of the `powerSpectrumPrimordialClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
 class  (powerSpectrumPrimordialClass), intent(inout)           :: self
 logical                              , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (powerSpectrumPrimordialClass), intent(inout)           :: self
 type   (inputParameters             ), intent(inout)           :: descriptor
 logical                              , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(powerSpectrumPrimordialClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (powerSpectrumPrimordialClass), intent(inout) :: self
 integer                              , intent(in   ) :: stateFile
 type   (fgsl_file                    ), intent(in   ) :: fgslStateFile
 integer(c_size_t                     ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`logarithmicDerivative` Return the logarithmic derivative with respect to wavenumber of the primordial power spectrum at the given `wavenumber` (specified in units of Mpc^{-1}. Must have the following interface:

```
double precision function myImplementationLogarithmicDerivative(self,wavenumber)
 class           (powerSpectrumPrimordialClass), intent(inout) :: self
 double precision                              , intent(in   ) :: wavenumber
end double precision function myImplementationLogarithmicDerivative
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (powerSpectrumPrimordialClass)                          , intent(inout) :: &
&  self
 type     (varying_string             ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                      )                          , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(powerSpectrumPrimordialClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(powerSpectrumPrimordialClass), intent(inout) :: self
 class(powerSpectrumPrimordialClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`power` Return the (unnormalized) power in the primordial power spectrum at the given `wavenumber` (specified in units of Mpc^{-1}. Must have the following interface:

```
double precision function myImplementationPower(self,wavenumber)
 class           (powerSpectrumPrimordialClass), intent(inout) :: self
 double precision                              , intent(in   ) :: wavenumber
end double precision function myImplementationPower
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (powerSpectrumPrimordialClass), intent(inout) :: self
 integer                              , intent(in   ) :: stateFile
 type   (fgsl_file                    ), intent(in   ) :: fgslStateFile
 integer(c_size_t                     ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`powerSpectrumPrimordialPowerLaw` Implements a power-law primordial power spectrum, possibly with a running index. It is defined by

$$P(k) \propto k^{n_\mathrm{s}+\ln(k/k_\mathrm{ref})[\mathrm{d}n/\mathrm{d}\ln k]}, \tag{16.9}$$

where the parameters are specified by input parameters $n_\mathrm{s} \equiv$ `[index]`, $k_\mathrm{ref} \equiv$ `[wavenumberReference]` and $\mathrm{d}n/\mathrm{d}\ln k \equiv$ `[running]`.

Transferred Primordial Power Spectrum

Additional implementations for transferred primordial power spectrum are added using the `powerSpectrumPrimordialTransf` class. The implementation should be placed in a file containing the directive:

```
!# <powerSpectrumPrimordialTransferred name="powerSpectrumPrimordialTransferredMyImplementation">
!# <description>A short description of the implementation.</description>
!# </powerSpectrumPrimordialTransferred>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `powerSpectrumPrimordialTransferredMyImplementation`. The file *must* define a type that extends the `powerSpectrumPrimordialTransferredClass` class (or extends another type which is itself an extension of the `powerSpectrumPrimordialTransferredClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (powerSpectrumPrimordialTransferredClass), intent(inout)           :: self
  logical                                         , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationDescriptor(self,descriptor,includeMethod)
  class  (powerSpectrumPrimordialTransferredClass), intent(inout)           :: self
  type   (inputParameters                        ), intent(inout)           :: &
&  descriptor
  logical                                         , intent(in   ), optional :: &
&  includeMethod
 end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAutoHook(self)
  class(powerSpectrumPrimordialTransferredClass), intent(inout) :: self
 end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (powerSpectrumPrimordialTransferredClass), intent(inout) :: self
 integer                                         , intent(in   ) :: stateFile
 type   (fgsl_file                               ), intent(in   ) :: fgslStateFile
 integer(c_size_t                                ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`logarithmicDerivative` Return the logarithmic derivative with respect to wavenumber of the transferred primordial power spectrum at the given `wavenumber` (specified in units of Mpc^{-1}. Must have the following interface:

```
double precision function myImplementationLogarithmicDerivative(self,wavenumber)
 class           (powerSpectrumPrimordialTransferredClass), intent(inout) :: self
 double precision                                         , intent(in   ) :: wavenumber
end double precision function myImplementationLogarithmicDerivative
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (powerSpectrumPrimordialTransferredClass)                          , intent(&
&  inout) :: self
 type     (varying_string                        ), allocatable, dimension(:), intent(&
&  inout) :: allowedParameters
 character(len=*                                 )                          , intent(&
&  in   ) :: sourceName
 end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(powerSpectrumPrimordialTransferredClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(powerSpectrumPrimordialTransferredClass), intent(inout) :: self
 class(powerSpectrumPrimordialTransferredClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`power` Return the (unnormalized) power in the transferred primordial power spectrum at the given `wavenumber` (specified in units of Mpc^{-1}. Must have the following interface:

```
double precision function myImplementationPower(self,wavenumber)
 class           (powerSpectrumPrimordialTransferredClass), intent(inout) :: self
 double precision                                         , intent(in   ) :: wavenumber
end double precision function myImplementationPower
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (powerSpectrumPrimordialTransferredClass), intent(inout) :: self
 integer                                         , intent(in   ) :: stateFile
 type   (fgsl_file                               ), intent(in   ) :: fgslStateFile
 integer(c_size_t                                ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`powerSpectrumPrimordialTransferredSimple` Implements a simple transferred primordial power spectrum.

Power Spectrum Window Functions

Additional implementations for power spectrum window functions are added using the `powerSpectrumWindowFunction` class. The implementation should be placed in a file containing the directive:

```
!# <powerSpectrumWindowFunction name="powerSpectrumWindowFunctionMyImplementation">
!# <description>A short description of the implementation.</description>
!# </powerSpectrumWindowFunction>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `powerSpectrumWindowFunctionMyImplementation`. The file *must* define a type that extends the `powerSpectrumWindowFunctionClass` class (or extends another type which is itself an extension of the `powerSpectrumWindowFunctionClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (powerSpectrumWindowFunctionClass), intent(inout)          :: self
  logical                                  , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (powerSpectrumWindowFunctionClass), intent(inout)          :: self
 type   (inputParameters                 ), intent(inout)          :: descriptor
 logical                                  , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(powerSpectrumWindowFunctionClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`value` Returns the window function for power spectrum variance computation at the specified `wavenumber` (in Mpc^{-1}) for a given `smoothingMass` (in $M_\odot$). Must have the following interface:

```
double precision function myImplementationValue(self,wavenumber, smoothingMass)
 class           (powerSpectrumWindowFunctionClass), intent(inout) :: self
 double precision                                  , intent(in   ) :: wavenumber, &
&  smoothingMass
 end double precision function myImplementationValue
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (powerSpectrumWindowFunctionClass), intent(inout) :: self
 integer                                  , intent(in   ) :: stateFile
 type   (fgsl_file                        ), intent(in   ) :: fgslStateFile
 integer(c_size_t                         ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`amplitudeIsMassIndependent` Should return true if, and only if, the amplitude of the window function below the maximum wavenumber is independent of the smoothing mass scale. A default implementation exists. If overridden the following interface must be used:

```
logical function myImplementationAmplitudeIsMassIndependent(self)
 class(powerSpectrumWindowFunctionClass), intent(inout) :: self
end logical function myImplementationAmplitudeIsMassIndependent
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (powerSpectrumWindowFunctionClass)                          , intent(inout) &
&  :: self
 type     (varying_string                 ), allocatable, dimension(:), intent(inout) &
&  :: allowedParameters
 character(len=*                          )                          , intent(in   ) &
&  :: sourceName
 end subroutine myImplementationAllowedParameters
```

`wavenumberMaximum` Returns the maximum wavenumber for which the window function for power spectrum variance computation is non-zero for a given `smoothingMass` (in $M_\odot$). Must have the following interface:

```
double precision function myImplementationWavenumberMaximum(self,smoothingMass)
 class           (powerSpectrumWindowFunctionClass), intent(inout) :: self
 double precision                                  , intent(in   ) :: smoothingMass
end double precision function myImplementationWavenumberMaximum
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(powerSpectrumWindowFunctionClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(powerSpectrumWindowFunctionClass), intent(inout) :: self
 class(powerSpectrumWindowFunctionClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (powerSpectrumWindowFunctionClass), intent(inout) :: self
 integer                                  , intent(in   ) :: stateFile
 type   (fgsl_file                        ), intent(in   ) :: fgslStateFile
 integer(c_size_t                         ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`powerSpectrumWindowFunctionTopHat` A top-hat in real space window function for filtering of power spectra.

`powerSpectrumWindowFunctionSharpKSpace` A sharp k-space window function for filtering of power spectra.

`powerSpectrumWindowFunctionLagrangianChan2017` A power spectrum window function class that implements the Lagrangian filter of Chan et al. [2017].

`powerSpectrumWindowFunctionTopHatSharpKHybrid` A hybrid top-hat/sharp k-space window function for filtering of power spectra.

`powerSpectrumWindowFunctionSmoothKSpace` A smooth window function for filtering of power spectra.

Radiation Fields

Additional implementations for radiation fields are added using the `radiationField` class. The implementation should be placed in a file containing the directive:

```
!# <radiationField name="radiationFieldMyImplementation">
!# <description>A short description of the implementation.</description>
!# </radiationField>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `radiationFieldMyImplementation`. The file *must* define a type that extends the `radiationFieldClass` class (or

extends another type which is itself an extension of the `radiationFieldClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
 class  (radiationFieldClass), intent(inout)           :: self
 logical                     , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (radiationFieldClass), intent(inout)           :: self
 type   (inputParameters    ), intent(inout)           :: descriptor
 logical                     , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(radiationFieldClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (radiationFieldClass), intent(inout) :: self
 integer                     , intent(in   ) :: stateFile
 type   (fgsl_file          ), intent(in   ) :: fgslStateFile
 integer(c_size_t           ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`flux` Return the flux (in units of ergs cm^2 s^{-1} Hz^{-1} ster^{-1}) of the given radiation field. Must have the following interface:

```
double precision function myImplementationFlux(self,wavelength,node)
 class           (radiationFieldClass), intent(inout) :: self
 double precision                     , intent(in   ) :: wavelength
 type            (treeNode           ), intent(inout) :: node
end double precision function myImplementationFlux
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (radiationFieldClass)                          , intent(inout) :: self
```

```
  type     (varying_string     ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
  character(len=*              )                           , intent(in   ) :: sourceName
 end subroutine myImplementationAllowedParameters
```

`integrateOverCrossSection` Integrates the flux (in units of ergs cm^2 s^{-1} Hz^{-1} ster^{-1}) of the given radiation structure between the wavelengths given in `wavelengthRange` over a cross section specified by the function `crossSectionFunction`. A default implementation exists. If overridden the following interface must be used:

```
 double precision function myImplementationIntegrateOverCrossSection(self,&
&  wavelengthRange,crossSectionFunction,node)
  class           (radiationFieldClass)              , intent(inout) :: self
  double precision                     , dimension(2), intent(in   ) :: wavelengthRange
  double precision                     , external                    :: &
&  crossSectionFunction
  type            (treeNode           )              , intent(inout) :: node
 end double precision function myImplementationIntegrateOverCrossSection
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(radiationFieldClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(radiationFieldClass), intent(inout) :: self
 class(radiationFieldClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (radiationFieldClass), intent(inout) :: self
 integer                     , intent(in   ) :: stateFile
 type   (fgsl_file          ), intent(in   ) :: fgslStateFile
 integer(c_size_t           ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`radiationFieldNull` A radiation field class for null fields.

`radiationFieldIntergalacticBackground` A radiation field class for intergalactic background light with properties read from file.

`radiationFieldIntergalacticBackgroundInternal` A radiation field class for intergalactic background light with properties computed internally.

`radiationFieldIntergalacticBackgroundFile` A radiation field class for intergalactic background light with properties read from file.

`radiationFieldCosmicMicrowaveBackground` A radiation field class for the cosmic microwave background.

`radiationFieldSummation` A summation radiation field class.

`radiationFieldBlackBody` A radiation field class for blackbody fields.

Ram pressure stripping in disks

Additional implementations for ram pressure stripping in disks are added using the `ramPressureStrippingDisks` class. The implementation should be placed in a file containing the directive:

```
!# <ramPressureStrippingDisks name="ramPressureStrippingDisksMyImplementation">
!# <description>A short description of the implementation.</description>
!# </ramPressureStrippingDisks>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `ramPressureStrippingDisksMyImplementation`. The file *must* define a type that extends the `ramPressureStrippingDisksClass` class (or extends another type which is itself an extension of the `ramPressureStrippingDisksClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (ramPressureStrippingDisksClass), intent(inout)           :: self
  logical                                , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (ramPressureStrippingDisksClass), intent(inout)           :: self
 type   (inputParameters               ), intent(inout)           :: descriptor
 logical                                , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`rateMassLoss` Returns the rate of mass loss (in $M_\odot$ Gyr^{-1}) due to ram pressure stripping of the disk component of `node`. Must have the following interface:

```
double precision function myImplementationRateMassLoss(self,node)
 class(ramPressureStrippingDisksClass), intent(inout) :: self
 type (treeNode                      ), intent(inout) :: node
end double precision function myImplementationRateMassLoss
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(ramPressureStrippingDisksClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (ramPressureStrippingDisksClass), intent(inout) :: self
 integer                                , intent(in   ) :: stateFile
 type   (fgsl_file                      ), intent(in   ) :: fgslStateFile
 integer(c_size_t                       ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (ramPressureStrippingDisksClass)                          , intent(inout) ::&
&  self
 type     (varying_string               ), allocatable, dimension(:), intent(inout) ::&
&  allowedParameters
 character(len=*                         )                          , intent(in   ) ::&
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(ramPressureStrippingDisksClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(ramPressureStrippingDisksClass), intent(inout) :: self
 class(ramPressureStrippingDisksClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (ramPressureStrippingDisksClass), intent(inout) :: self
 integer                                , intent(in   ) :: stateFile
 type   (fgsl_file                      ), intent(in   ) :: fgslStateFile
 integer(c_size_t                       ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`ramPressureStrippingDisksNull` A null model of ram pressure stripping in galactic disks.

`ramPressureStrippingDisksSimple` A simple model of ram pressure stripping in galactic disks.

Ram pressure stripping in spheroids

Additional implementations for ram pressure stripping in spheroids are added using the `ramPressureStrippingSpheroids` class. The implementation should be placed in a file containing the directive:

```
!# <ramPressureStrippingSpheroids name="ramPressureStrippingSpheroidsMyImplementation">
!# <description>A short description of the implementation.</description>
!# </ramPressureStrippingSpheroids>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `ramPressureStrippingSpheroidsMyImplementation`. The file *must* define a type that extends the `ramPressureStrippingSpheroidsClass` class (or extends another type which is itself an extension of the `ramPressureStrippingSpheroidsClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (ramPressureStrippingSpheroidsClass), intent(inout)           :: self
  logical                                    , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (ramPressureStrippingSpheroidsClass), intent(inout)           :: self
 type   (inputParameters                   ), intent(inout)           :: descriptor
 logical                                    , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`rateMassLoss` Returns the rate of mass loss (in $M_\odot$ Gyr^{-1}) due to ram pressure stripping of the spheroid component of `node`. Must have the following interface:

```
double precision function myImplementationRateMassLoss(self,node)
 class(ramPressureStrippingSpheroidsClass), intent(inout) :: self
 type (treeNode                          ), intent(inout) :: node
end double precision function myImplementationRateMassLoss
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(ramPressureStrippingSpheroidsClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (ramPressureStrippingSpheroidsClass), intent(inout) :: self
 integer                                    , intent(in   ) :: stateFile
 type   (fgsl_file                         ), intent(in   ) :: fgslStateFile
 integer(c_size_t                          ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (ramPressureStrippingSpheroidsClass)                          , intent(inout&
&  ) :: self
 type     (varying_string                    ), allocatable, dimension(:), intent(inout&
&  ) :: allowedParameters
 character(len=*                             )                          , intent(in   &
&  ) :: sourceName
 end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(ramPressureStrippingSpheroidsClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(ramPressureStrippingSpheroidsClass), intent(inout) :: self
 class(ramPressureStrippingSpheroidsClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (ramPressureStrippingSpheroidsClass), intent(inout) :: self
 integer                                    , intent(in   ) :: stateFile
 type   (fgsl_file                         ), intent(in   ) :: fgslStateFile
 integer(c_size_t                          ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`ramPressureStrippingSpheroidsSimple` A simple model of ram pressure stripping in galactic spheroids.

`ramPressureStrippingSpheroidsNull` A null model of ram pressure stripping in galactic spheroids.

Dynamical friction models.

Additional implementations for dynamical friction models. are added using the `satelliteDynamicalFriction` class. The implementation should be placed in a file containing the directive:

```
!# <satelliteDynamicalFriction name="satelliteDynamicalFrictionMyImplementation">
!# <description>A short description of the implementation.</description>
!# </satelliteDynamicalFriction>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `satelliteDynamicalFrictionMyImplementation`. The file *must* define a type that extends the `satelliteDynamicalFrictionClass` class (or extends another type which is itself an extension of the `satelliteDynamicalFrictionClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (satelliteDynamicalFrictionClass), intent(inout)           :: self
  logical                                 , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (satelliteDynamicalFrictionClass), intent(inout)           :: self
 type   (inputParameters                ), intent(inout)           :: descriptor
 logical                                 , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(satelliteDynamicalFrictionClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`acceleration` Returns the satellite acceleration due to dynamical friction for `node` (in units of km/s/-Gyr). Must have the following interface:

```
double precision, dimension(3) function myImplementationAcceleration(self,node)
 class(satelliteDynamicalFrictionClass), intent(inout) :: self
 type (treeNode                       ), intent(inout) :: node
end double precision, dimension(3) function myImplementationAcceleration
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (satelliteDynamicalFrictionClass), intent(inout) :: self
 integer                                 , intent(in   ) :: stateFile
 type   (fgsl_file                      ), intent(in   ) :: fgslStateFile
 integer(c_size_t                       ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (satelliteDynamicalFrictionClass)                          , intent(inout) &
&  :: self
 type     (varying_string                ), allocatable, dimension(:), intent(inout) &
&  :: allowedParameters
 character(len=*                         )                          , intent(in   ) &
&  :: sourceName
 end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(satelliteDynamicalFrictionClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(satelliteDynamicalFrictionClass), intent(inout) :: self
 class(satelliteDynamicalFrictionClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (satelliteDynamicalFrictionClass), intent(inout) :: self
 integer                                 , intent(in   ) :: stateFile
 type   (fgsl_file                      ), intent(in   ) :: fgslStateFile
 integer(c_size_t                       ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`satelliteDynamicalFrictionChandrasekhar1943` A satellite dynamical friction class which uses the model of Chandrasekhar [1943].

`satelliteDynamicalFrictionZero` A satellite dynamical friction class in which the acceleration is always zero.

Satellite Merging Timescales

Additional implementations for satellite merging timescales are added using the `satelliteMergingTimescales` class. The implementation should be placed in a file containing the directive:

```
!# <satelliteMergingTimescales name="satelliteMergingTimescalesMyImplementation">
!# <description>A short description of the implementation.</description>
!# </satelliteMergingTimescales>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `satelliteMergingTimescalesMyImplementation`. The file *must* define a type that extends the `satelliteMergingTimescalesClass` class (or extends another type which is itself an extension of the `satelliteMergingTimescalesClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (satelliteMergingTimescalesClass), intent(inout)          :: self
  logical                                 , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (satelliteMergingTimescalesClass), intent(inout)          :: self
 type   (inputParameters                ), intent(inout)          :: descriptor
 logical                                 , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(satelliteMergingTimescalesClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (satelliteMergingTimescalesClass), intent(inout) :: self
 integer                                 , intent(in   ) :: stateFile
 type   (fgsl_file                       ), intent(in   ) :: fgslStateFile
 integer(c_size_t                        ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (satelliteMergingTimescalesClass)                          , intent(inout) &
&  :: self
 type     (varying_string                ), allocatable, dimension(:), intent(inout) &
&  :: allowedParameters
 character(len=*                         )                          , intent(in   ) &
&  :: sourceName
end subroutine myImplementationAllowedParameters
```

`timeUntilMerging` Return the time (in Gyr) until the satellite will merge with its host given the current orbit. Must have the following interface:

```
double precision function myImplementationTimeUntilMerging(self,node,orbit)
 class(satelliteMergingTimescalesClass), intent(inout) :: self
 type (treeNode                       ), intent(inout) :: node
 type (keplerOrbit                    ), intent(inout) :: orbit
end double precision function myImplementationTimeUntilMerging
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(satelliteMergingTimescalesClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(satelliteMergingTimescalesClass), intent(inout) :: self
 class(satelliteMergingTimescalesClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (satelliteMergingTimescalesClass), intent(inout) :: self
 integer                                 , intent(in   ) :: stateFile
 type   (fgsl_file                       ), intent(in   ) :: fgslStateFile
 integer(c_size_t                        ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`satelliteMergingTimescalesLaceyCole1993Tormen` Computes the merging timescale using the method of Lacey and Cole [1993] with a parameterization of orbital parameters designed to fit the results of Tormen [1997] as described by Cole et al. [2000]. See §12.51.1.

`satelliteMergingTimescalesZero` Returns a zero timescale for merging.

`satelliteMergingTimescalesVillalobos2013` Computes the merging timescale using the method of Villalobos et al. [2013] to modify another merging timescale method. See §12.51.1.

`satelliteMergingTimescalesWetzelWhite2010` Computes the merging timescale using the method of Wetzel and White [2010]. See §12.51.1.

`satelliteMergingTimescalesBoylanKolchin2008` Computes the merging timescale using the method of Boylan-Kolchin et al. [2008]. See §12.51.1.

`satelliteMergingTimescalesPreset` This class assumes that merging times have been preset for every node (or, at least, every node which becomes a satellite). It therefore simply returns the preset merging time. See §12.51.1.

`satelliteMergingTimescalesRandom` Returns a random timescale for merging.

`satelliteMergingTimescalesJiang2008` Computes the merging timescale using the method of Jiang et al. [2008]. See §12.51.1.

`satelliteMergingTimescalesInfinite` Returns an infinite timescale for merging. See §12.51.1.

`satelliteMergingTimescalesLaceyCole1993` Computes the merging timescale using the method of Lacey and Cole [1993]. See §12.51.1.

Satellite Orphan Distributions

Additional implementations for satellite orphan distributions are added using the `satelliteOrphanDistribution` class. The implementation should be placed in a file containing the directive:

```
!# <satelliteOrphanDistribution name="satelliteOrphanDistributionMyImplementation">
!# <description>A short description of the implementation.</description>
!# </satelliteOrphanDistribution>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `satelliteOrphanDistributionMyImplementation`. The file *must* define a type that extends the `satelliteOrphanDistributionClass` class (or extends another type which is itself an extension of the `satelliteOrphanDistributionClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (satelliteOrphanDistributionClass), intent(inout)          :: self
  logical                                  , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (satelliteOrphanDistributionClass), intent(inout)          :: self
 type   (inputParameters                ), intent(inout)          :: descriptor
 logical                                 , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(satelliteOrphanDistributionClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (satelliteOrphanDistributionClass), intent(inout) :: self
 integer                                  , intent(in   ) :: stateFile
 type   (fgsl_file                       ), intent(in   ) :: fgslStateFile
 integer(c_size_t                        ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`position` Return the position of the given orphan in physical coordinates. Must have the following interface:

```
double precision, dimension(3) function myImplementationPosition(self,node)
 class(satelliteOrphanDistributionClass), intent(inout) :: self
 type (treeNode                        ), intent(inout) :: node
end double precision, dimension(3) function myImplementationPosition
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (satelliteOrphanDistributionClass)                         , intent(inout) &
&  :: self
 type     (varying_string                 ), allocatable, dimension(:), intent(inout) &
&  :: allowedParameters
 character(len=*                          )                          , intent(in   ) &
&  :: sourceName
 end subroutine myImplementationAllowedParameters
```

`velocity` Return the peculiar velocity of the given orphan in physical coordinates. Must have the following interface:

```
double precision, dimension(3) function myImplementationVelocity(self,node)
 class(satelliteOrphanDistributionClass), intent(inout) :: self
 type (treeNode                        ), intent(inout) :: node
end double precision, dimension(3) function myImplementationVelocity
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(satelliteOrphanDistributionClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(satelliteOrphanDistributionClass), intent(inout) :: self
 class(satelliteOrphanDistributionClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (satelliteOrphanDistributionClass), intent(inout) :: self
 integer                                  , intent(in   ) :: stateFile
 type   (fgsl_file                       ), intent(in   ) :: fgslStateFile
 integer(c_size_t                        ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

`extent` The maximum extent of the distribution, i.e. the radius of a sphere centered on the host halo which encompasses all orphan satellites. Must have the following interface:

```
double precision function myImplementationExtent(self,node)
 class(satelliteOrphanDistributionClass), intent(inout) :: self
 type (treeNode                         ), intent(inout) :: node
end double precision function myImplementationExtent
```

Existing implementations are:

`satelliteOrphanDistributionTraceDarkMatter` An orphan satellite distribution which assumes an isotropic, random distribution with orphans tracing the radial distribution of dark matter.

`satelliteOrphanDistributionRandomIsotropic` An abstract orphan satellite distribution which assumes an isotropic, random distribution of positions, and velocities drawn from an isotropic normal distribution. The radial distribution and velocity dispersion must be specified by the child class.

Satellite halo tidal field models.

Additional implementations for satellite halo tidal field models. are added using the `satelliteTidalField` class. The implementation should be placed in a file containing the directive:

```
!# <satelliteTidalField name="satelliteTidalFieldMyImplementation">
!# <description>A short description of the implementation.</description>
!# </satelliteTidalField>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `satelliteTidalFieldMyImplementation`. The file *must* define a type that extends the `satelliteTidalFieldClass` class (or extends another type which is itself an extension of the `satelliteTidalFieldClass` class),

containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`tidalTensorRadial` Returns the radial component, $\Phi_{\rm rr}$, of the tidal tensor, $\Phi_{\rm ab}$. Must have the following interface:

```
double precision function myImplementationTidalTensorRadial(self,node)
 class(satelliteTidalFieldClass), intent(inout) :: self
 type (treeNode                ), intent(inout) :: node
end double precision function myImplementationTidalTensorRadial
```

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
 class  (satelliteTidalFieldClass), intent(inout)           :: self
 logical                          , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (satelliteTidalFieldClass), intent(inout)           :: self
 type   (inputParameters         ), intent(inout)           :: descriptor
 logical                          , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(satelliteTidalFieldClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (satelliteTidalFieldClass), intent(inout) :: self
 integer                          , intent(in   ) :: stateFile
 type   (fgsl_file                ), intent(in   ) :: fgslStateFile
 integer(c_size_t                 ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (satelliteTidalFieldClass)                          , intent(inout) :: self
 type     (varying_string          ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
```

```
  character(len=*                  )                          , intent(in   ) :: &
 &  sourceName
 end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(satelliteTidalFieldClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(satelliteTidalFieldClass), intent(inout) :: self
 class(satelliteTidalFieldClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (satelliteTidalFieldClass), intent(inout) :: self
 integer                          , intent(in   ) :: stateFile
 type   (fgsl_file               ), intent(in   ) :: fgslStateFile
 integer(c_size_t                ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`satelliteTidalFieldNull` A satellite tidal field class which computes the tidal field assuming no tidal field.

`satelliteTidalFieldSphericalSymmetry` A satellite tidal field class which computes the tidal field assuming spherical symmetry.

Satellite halo tidal heating rate models.

Additional implementations for satellite halo tidal heating rate models. are added using the `satelliteTidalHeatingRate` class. The implementation should be placed in a file containing the directive:

```
!# <satelliteTidalHeatingRate name="satelliteTidalHeatingRateMyImplementation">
!# <description>A short description of the implementation.</description>
!# </satelliteTidalHeatingRate>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `satelliteTidalHeatingRateMyImplementation`. The file *must* define a type that extends the `satelliteTidalHeatingRateClass` class (or extends another type which is itself an extension of the `satelliteTidalHeatingRateClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`heatingRate` Return the satellite tidal heating rate for `node` (in units of $(\mathrm{km/s/Mpc})^2/\mathrm{Gyr}$). Must have the following interface:

```
double precision function myImplementationHeatingRate(self,node)
 class(satelliteTidalHeatingRateClass), intent(inout) :: self
 type (treeNode                      ), intent(inout) :: node
end double precision function myImplementationHeatingRate
```

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (satelliteTidalHeatingRateClass), intent(inout)           :: self
  logical                                , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (satelliteTidalHeatingRateClass), intent(inout)           :: self
 type   (inputParameters               ), intent(inout)           :: descriptor
 logical                                , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(satelliteTidalHeatingRateClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (satelliteTidalHeatingRateClass), intent(inout) :: self
 integer                                , intent(in   ) :: stateFile
 type   (fgsl_file                      ), intent(in   ) :: fgslStateFile
 integer(c_size_t                       ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (satelliteTidalHeatingRateClass)                          , intent(inout) ::&
&   self
 type     (varying_string                ), allocatable, dimension(:), intent(inout) ::&
&   allowedParameters
 character(len=*                         )                          , intent(in   ) ::&
&   sourceName
 end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(satelliteTidalHeatingRateClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(satelliteTidalHeatingRateClass), intent(inout) :: self
 class(satelliteTidalHeatingRateClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (satelliteTidalHeatingRateClass), intent(inout) :: self
 integer                                , intent(in   ) :: stateFile
 type   (fgsl_file                      ), intent(in   ) :: fgslStateFile
 integer(c_size_t                       ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`satelliteTidalHeatingRateZero` A satellite tidal heating rate class which implements a tidal heating rate model in which the heating rate is always zero.

`satelliteTidalHeatingRateGnedin1999` A satellite tidal heating rate class which implements the tidal heating rate model of Gnedin et al. [1999].

Tidal stripping models for satellites.

Additional implementations for tidal stripping models for satellites. are added using the `satelliteTidalStripping` class. The implementation should be placed in a file containing the directive:

```
!# <satelliteTidalStripping name="satelliteTidalStrippingMyImplementation">
!# <description>A short description of the implementation.</description>
!# </satelliteTidalStripping>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `satelliteTidalStrippingMyImplementation`. The file *must* define a type that extends the `satelliteTidalStrippingClass` class (or extends another type which is itself an extension of the `satelliteTidalStrippingClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (satelliteTidalStrippingClass), intent(inout)          :: self
  logical                              , intent(in   ), optional :: includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (satelliteTidalStrippingClass), intent(inout)          :: self
 type   (inputParameters             ), intent(inout)          :: descriptor
 logical                              , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(satelliteTidalStrippingClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (satelliteTidalStrippingClass), intent(inout) :: self
 integer                              , intent(in   ) :: stateFile
 type   (fgsl_file                    ), intent(in   ) :: fgslStateFile
 integer(c_size_t                     ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (satelliteTidalStrippingClass)                          , intent(inout) :: &
&  self
 type     (varying_string              ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                       )                          , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`massLossRate` Returns the rate of tidal mass loss for `node` (in units of $M_\odot$/Gyr). Must have the following interface:

```
double precision function myImplementationMassLossRate(self,node)
 class(satelliteTidalStrippingClass), intent(inout)         :: self
 type (treeNode                    ), intent(inout), target :: node
end double precision function myImplementationMassLossRate
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(satelliteTidalStrippingClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(satelliteTidalStrippingClass), intent(inout) :: self
 class(satelliteTidalStrippingClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (satelliteTidalStrippingClass), intent(inout) :: self
 integer                              , intent(in   ) :: stateFile
 type   (fgsl_file                    ), intent(in   ) :: fgslStateFile
 integer(c_size_t                     ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`satelliteTidalStrippingZero` A satellite tidal stripping class in which the stripping rate is always zero.

`satelliteTidalStrippingZentner2005` A satellite tidal stripping class which follows the model of Zentner et al. [2005].

Expulsive feedback from star formation in disks

Additional implementations for expulsive feedback from star formation in disks are added using the `starFormationExpulsiveFeedbackDisks` class. The implementation should be placed in a file containing the directive:

```
!# <starFormationExpulsiveFeedbackDisks name="starFormationExpulsiveFeedbackDisksMyImplementation">
!# <description>A short description of the implementation.</description>
!# </starFormationExpulsiveFeedbackDisks>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `starFormationExpulsiveFeedbackDisksMyImplementation`. The file *must* define a type that extends the `starFormationExpulsiveFeedbackDisksClass` class (or extends another type which is itself an extension of the `starFormationExpulsiveFeedbackDisksClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (starFormationExpulsiveFeedbackDisksClass), intent(inout)          :: self
  logical                                          , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationDescriptor(self,descriptor,includeMethod)
  class  (starFormationExpulsiveFeedbackDisksClass), intent(inout)          :: self
  type   (inputParameters                         ), intent(inout)          :: &
&  descriptor
  logical                                          , intent(in   ), optional :: &
&  includeMethod
 end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAutoHook(self)
  class(starFormationExpulsiveFeedbackDisksClass), intent(inout) :: self
 end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
  class  (starFormationExpulsiveFeedbackDisksClass), intent(inout) :: self
  integer                                          , intent(in   ) :: stateFile
  type   (fgsl_file                                ), intent(in   ) :: fgslStateFile
  integer(c_size_t                                 ), intent(in   ) :: stateOperationID
 end subroutine myImplementationStateStore
```

`outflowRate` Returns the outflow rate due to star formation in the disk component of `node` in units of $M_\odot$/Gyr. Must have the following interface:

```
 double precision function myImplementationOutflowRate(self,node,rateEnergyInput, &
&  rateStarFormation)
  class           (starFormationExpulsiveFeedbackDisksClass), intent(inout) :: self
  type            (treeNode                                ), intent(inout) :: node
  double precision                                          , intent(in   ) :: &
&  rateEnergyInput, rateStarFormation
 end double precision function myImplementationOutflowRate
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
  class    (starFormationExpulsiveFeedbackDisksClass)                          , intent&
&  (inout) :: self
  type     (varying_string                          ), allocatable, dimension(:), intent&
&  (inout) :: allowedParameters
```

```
     character(len=*                                    )                          , intent&
    &  (in   ) :: sourceName
     end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(starFormationExpulsiveFeedbackDisksClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(starFormationExpulsiveFeedbackDisksClass), intent(inout) :: self
 class(starFormationExpulsiveFeedbackDisksClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (starFormationExpulsiveFeedbackDisksClass), intent(inout) :: self
 integer                                          , intent(in   ) :: stateFile
 type   (fgsl_file                                ), intent(in   ) :: fgslStateFile
 integer(c_size_t                                 ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`starFormationExpulsiveFeedbackDisksZero` A zero expulsive outflow rate due to star formation feedback in galactic disks.

`starFormationExpulsiveFeedbackDisksSuperWind` A superwind expulsive outflow rate due to star formation feedback in galactic disks.

Epulsive feedback from star formation in spheroids

Additional implementations for epulsive feedback from star formation in spheroids are added using the `starFormationExpulsiveFeedbackSpheroids` class. The implementation should be placed in a file containing the directive:

```
!# <starFormationExpulsiveFeedbackSpheroids name="starFormationExpulsiveFeedbackSpheroidsMyImplementatio
!# <description>A short description of the implementation.</description>
!# </starFormationExpulsiveFeedbackSpheroids>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `starFormationExpulsiveFeedbackSpheroidsMyImplementation`. The file *must* define a type that extends the `starFormationExpulsiveFeedbackSpheroidsClass` class (or extends another type which is itself an extension of the `starFormationExpulsiveFeedbackSpheroidsClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (starFormationExpulsiveFeedbackSpheroidsClass), intent(inout)           :: self
  logical                                              , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationDescriptor(self,descriptor,includeMethod)
  class  (starFormationExpulsiveFeedbackSpheroidsClass), intent(inout)           :: self
  type   (inputParameters                              ), intent(inout)           :: &
&  descriptor
  logical                                              , intent(in   ), optional :: &
&  includeMethod
 end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAutoHook(self)
  class(starFormationExpulsiveFeedbackSpheroidsClass), intent(inout) :: self
 end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
  class  (starFormationExpulsiveFeedbackSpheroidsClass), intent(inout) :: self
  integer                                              , intent(in   ) :: stateFile
  type   (fgsl_file                                    ), intent(in   ) :: fgslStateFile
  integer(c_size_t                                     ), intent(in   ) :: &
&  stateOperationID
 end subroutine myImplementationStateStore
```

`outflowRate` Returns the outflow rate due to star formation in the spheroid component of `node` in units of $M_\odot$/Gyr. Must have the following interface:

```
 double precision function myImplementationOutflowRate(self,node,rateEnergyInput, &
&  rateStarFormation)
  class           (starFormationExpulsiveFeedbackSpheroidsClass), intent(inout) :: self
  type            (treeNode                                    ), intent(inout) :: node
  double precision                                              , intent(in   ) :: &
&  rateEnergyInput, rateStarFormation
 end double precision function myImplementationOutflowRate
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
  class    (starFormationExpulsiveFeedbackSpheroidsClass)                          , &
&  intent(inout) :: self
  type     (varying_string                              ), allocatable, dimension(:), &
&  intent(inout) :: allowedParameters
  character(len=*                                       )                          , &
&  intent(in   ) :: sourceName
 end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(starFormationExpulsiveFeedbackSpheroidsClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(starFormationExpulsiveFeedbackSpheroidsClass), intent(inout) :: self
 class(starFormationExpulsiveFeedbackSpheroidsClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (starFormationExpulsiveFeedbackSpheroidsClass), intent(inout) :: self
 integer                                              , intent(in   ) :: stateFile
 type   (fgsl_file                                    ), intent(in   ) :: fgslStateFile
 integer(c_size_t                                     ), intent(in   ) :: &
&  stateOperationID
 end subroutine myImplementationStateRestore
```

Existing implementations are:

`starFormationExpulsiveFeedbackSpheroidsSuperWind` A superwind expulsive outflow rate due to star formation feedback in galactic spheroids.

`starFormationExpulsiveFeedbackSpheroidsZero` A zero expulsive outflow rate due to star formation feedback in galactic spheroids.

Feedback from star formation in disks

Additional implementations for feedback from star formation in disks are added using the `starFormationFeedbackDisks` class. The implementation should be placed in a file containing the directive:

```
!# <starFormationFeedbackDisks name="starFormationFeedbackDisksMyImplementation">
!# <description>A short description of the implementation.</description>
!# </starFormationFeedbackDisks>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `starFormationFeedbackDisksMyImplementation`. The file *must* define a type that extends the `starFormationFeedbackDisksClass` class (or extends another type which is itself an extension of the `starFormationFeedbackDisksClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (starFormationFeedbackDisksClass), intent(inout)           :: self
  logical                                 , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (starFormationFeedbackDisksClass), intent(inout)           :: self
 type   (inputParameters                ), intent(inout)           :: descriptor
 logical                                 , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(starFormationFeedbackDisksClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (starFormationFeedbackDisksClass), intent(inout) :: self
 integer                                 , intent(in   ) :: stateFile
 type   (fgsl_file                      ), intent(in   ) :: fgslStateFile
 integer(c_size_t                       ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`outflowRate` Returns the outflow rate due to star formation in the disk component of `node` in units of $M_\odot$/Gyr. Must have the following interface:

```
 double precision function myImplementationOutflowRate(self,node,rateEnergyInput, &
&  rateStarFormation)
  class           (starFormationFeedbackDisksClass), intent(inout) :: self
  type            (treeNode                       ), intent(inout) :: node
```

```
  double precision                                , intent(in   ) :: rateEnergyInput, &
&  rateStarFormation
 end double precision function myImplementationOutflowRate
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
  class    (starFormationFeedbackDisksClass)                          , intent(inout) &
&  :: self
  type     (varying_string                ), allocatable, dimension(:), intent(inout) &
&  :: allowedParameters
  character(len=*                         )                          , intent(in   ) &
&  :: sourceName
 end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationObjectType(self)
  class(starFormationFeedbackDisksClass), intent(inout) :: self
 end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationDeepCopy(self,destination)
  class(starFormationFeedbackDisksClass), intent(inout) :: self
  class(starFormationFeedbackDisksClass), intent(inout) :: destination
 end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
  class  (starFormationFeedbackDisksClass), intent(inout) :: self
  integer                                 , intent(in   ) :: stateFile
  type   (fgsl_file                       ), intent(in   ) :: fgslStateFile
  integer(c_size_t                        ), intent(in   ) :: stateOperationID
 end subroutine myImplementationStateRestore
```

Existing implementations are:

`starFormationFeedbackDisksVlctyMxSclng` An outflow rate due to star formation feedback in galactic disks which scales with peak halo velocity.

`starFormationFeedbackDisksCreasey2012` The Creasey et al. [2012] outflow rate due to star formation feedback in galactic disks.

`starFormationFeedbackDisksHaloScaling` An outflow rate due to star formation feedback in galactic disks which scales with halo velocity.

`starFormationFeedbackDisksFixed` A fixed fraction outflow rate due to star formation feedback in galactic disks.

`starFormationFeedbackDisksPowerLawRedshiftScaling` A power-law outflow rate due to star formation feedback in galactic disks in which the characteristic velocity scales as a power of $(1+z)$.

`starFormationFeedbackDisksPowerLaw` A power-law outflow rate due to star formation feedback in galactic disks.

Feedback from star formation in spheroids

Additional implementations for feedback from star formation in spheroids are added using the `starFormationFeedbackSphero` class. The implementation should be placed in a file containing the directive:

```
!# <starFormationFeedbackSpheroids name="starFormationFeedbackSpheroidsMyImplementation">
!# <description>A short description of the implementation.</description>
!# </starFormationFeedbackSpheroids>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `starFormationFeedbackSpheroidsMyImplementation`. The file *must* define a type that extends the `starFormationFeedbackSpheroidsClass` class (or extends another type which is itself an extension of the `starFormationFeedbackSpheroidsClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (starFormationFeedbackSpheroidsClass), intent(inout)           :: self
  logical                                     , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (starFormationFeedbackSpheroidsClass), intent(inout)           :: self
 type   (inputParameters                    ), intent(inout)           :: descriptor
 logical                                     , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(starFormationFeedbackSpheroidsClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (starFormationFeedbackSpheroidsClass), intent(inout) :: self
 integer                                     , intent(in   ) :: stateFile
 type   (fgsl_file                          ), intent(in   ) :: fgslStateFile
 integer(c_size_t                           ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`outflowRate` Returns the outflow rate due to star formation in the spheroid component of `node` in units of $M_\odot$/Gyr. Must have the following interface:

```
double precision function myImplementationOutflowRate(self,node,rateEnergyInput, &
&  rateStarFormation)
 class           (starFormationFeedbackSpheroidsClass), intent(inout) :: self
 type            (treeNode                           ), intent(inout) :: node
 double precision                                     , intent(in   ) :: &
&  rateEnergyInput, rateStarFormation
 end double precision function myImplementationOutflowRate
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (starFormationFeedbackSpheroidsClass)                          , intent(&
&  inout) :: self
 type     (varying_string                     ), allocatable, dimension(:), intent(&
&  inout) :: allowedParameters
 character(len=*                              )                          , intent(in  &
&   ) :: sourceName
 end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(starFormationFeedbackSpheroidsClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(starFormationFeedbackSpheroidsClass), intent(inout) :: self
 class(starFormationFeedbackSpheroidsClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (starFormationFeedbackSpheroidsClass), intent(inout) :: self
 integer                                     , intent(in   ) :: stateFile
 type   (fgsl_file                          ), intent(in   ) :: fgslStateFile
 integer(c_size_t                           ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`starFormationFeedbackSpheroidsVlctyMxSclng` An outflow rate due to star formation feedback in galactic spheroids which scales with peak halo velocity.

`starFormationFeedbackSpheroidsPowerLawRedshiftScaling` A power-law outflow rate due to star formation feedback in galactic spheroids in which the characteristic velocity scales as a power of $(1+z)$.

`starFormationFeedbackSpheroidsPowerLaw` A power-law outflow rate due to star formation feedback in galactic spheroids.

`starFormationFeedbackSpheroidsFixed` A fixed fraction outflow rate due to star formation feedback in galactic spheroids.

Star Formation Histories

Additional implementations for star formation histories are added using the `starFormationHistory` class. The implementation should be placed in a file containing the directive:

```
!# <starFormationHistory name="starFormationHistoryMyImplementation">
!# <description>A short description of the implementation.</description>
!# </starFormationHistory>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `starFormationHistoryMyImplementation`. The file *must* define a type that extends the `starFormationHistoryClass` class (or extends another type which is itself an extension of the `starFormationHistoryClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`create` Create the star formation history object. Must have the following interface:

```
subroutine myImplementationCreate(self,node,historyStarFormation,timeBegin)
 class           (starFormationHistoryClass), intent(inout) :: self
 type            (treeNode                 ), intent(inout) :: node
 type            (history                  ), intent(inout) :: historyStarFormation
 double precision                            , intent(in   ) :: timeBegin
end subroutine myImplementationCreate
```

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
 class  (starFormationHistoryClass), intent(inout)          :: self
 logical                           , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (starFormationHistoryClass), intent(inout)          :: self
 type   (inputParameters          ), intent(inout)          :: descriptor
 logical                           , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(starFormationHistoryClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (starFormationHistoryClass), intent(inout) :: self
 integer                           , intent(in   ) :: stateFile
 type   (fgsl_file                 ), intent(in   ) :: fgslStateFile
 integer(c_size_t                  ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (starFormationHistoryClass)                          , intent(inout) :: self
 type     (varying_string           ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                    )                          , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`output` Output the star formation history. Must have the following interface:

```
 subroutine myImplementationOutput(self,node,nodePassesFilter,historyStarFormation,&
&  indexOutput,indexTree,labelComponent)
  class    (starFormationHistoryClass), intent(inout)         :: self
  type     (treeNode                 ), intent(inout), target :: node
  logical                             , intent(in   )         :: nodePassesFilter
  type     (history                  ), intent(inout)         :: historyStarFormation
  integer  (c_size_t                 ), intent(in   )         :: indexOutput
  integer  (kind=kind_int8           ), intent(in   )         :: indexTree
  character(len=*                    ), intent(in   )         :: labelComponent
 end subroutine myImplementationOutput
```

`rate` Record the rate of star formation in this history. Must have the following interface:

```
 subroutine myImplementationRate(self,node,historyStarFormation,abundancesFuel,&
&  rateStarFormation)
  class           (starFormationHistoryClass), intent(inout) :: self
  type            (treeNode                 ), intent(inout) :: node
  type            (history                  ), intent(inout) :: historyStarFormation
  type            (abundances               ), intent(in   ) :: abundancesFuel
```

```
 double precision                            , intent(in   ) :: rateStarFormation
end subroutine myImplementationRate
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(starFormationHistoryClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(starFormationHistoryClass), intent(inout) :: self
 class(starFormationHistoryClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (starFormationHistoryClass), intent(inout) :: self
 integer                           , intent(in   ) :: stateFile
 type   (fgsl_file                ), intent(in   ) :: fgslStateFile
 integer(c_size_t                 ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

`scales` Set ODE solver absolute scales for a star formation history object. Must have the following interface:

```
subroutine myImplementationScales(self,historyStarFormation,massStellar,&
&  abundancesStellar)
 class           (starFormationHistoryClass), intent(inout) :: self
 type            (history                  ), intent(inout) :: historyStarFormation
 double precision                           , intent(in   ) :: massStellar
 type            (abundances               ), intent(in   ) :: abundancesStellar
end subroutine myImplementationScales
```

Existing implementations are:

`starFormationHistoryMetallicitySplit` A star formation histories class which records star formation split by metallicity.

`starFormationHistoryNull` A null star formation histories class.

`starFormationHistoryInSitu` A star formation histories class which records *in situ* star formation.

Surface density rates of star formation in disks.

Additional implementations for surface density rates of star formation in disks. are added using the `starFormationRateSurfaceDensityDisks` class. The implementation should be placed in a file containing the directive:

```
!# <starFormationRateSurfaceDensityDisks name="starFormationRateSurfaceDensityDisksMyImplementation">
!# <description>A short description of the implementation.</description>
!# </starFormationRateSurfaceDensityDisks>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `starFormationRateSurfaceDensityDisksMyImplementation`. The file *must* define a type that extends the `starFormationRateSurfaceDensityDisksClass` class (or extends another type which is itself an extension of the `starFormationRateSurfaceDensityDisksClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (starFormationRateSurfaceDensityDisksClass), intent(inout)           :: self
  logical                                           , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationDescriptor(self,descriptor,includeMethod)
  class  (starFormationRateSurfaceDensityDisksClass), intent(inout)           :: self
  type   (inputParameters                           ), intent(inout)           :: &
&  descriptor
  logical                                            , intent(in   ), optional :: &
&  includeMethod
 end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAutoHook(self)
  class(starFormationRateSurfaceDensityDisksClass), intent(inout) :: self
 end subroutine myImplementationAutoHook
```

`unchanged` Return true if the surface density rate of star formation is unchanged since the previous evaluation. A default implementation exists. If overridden the following interface must be used:

```
 logical function myImplementationUnchanged(self,node)
  class(starFormationRateSurfaceDensityDisksClass), intent(inout) :: self
  type (treeNode                                 ), intent(inout) :: node
 end logical function myImplementationUnchanged
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (starFormationRateSurfaceDensityDisksClass), intent(inout) :: self
 integer                                           , intent(in   ) :: stateFile
 type   (fgsl_file                                 ), intent(in   ) :: fgslStateFile
 integer(c_size_t                                  ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (starFormationRateSurfaceDensityDisksClass)                          , &
&  intent(inout) :: self
 type     (varying_string                           ), allocatable, dimension(:), &
&  intent(inout) :: allowedParameters
 character(len=*                                    )                          , &
&  intent(in   ) :: sourceName
 end subroutine myImplementationAllowedParameters
```

`rate` Returns the star formation rate surface density (in $M_\odot$ Gyr^{-1} Mpc^{-2}) in the disk component of `thisNode` at the given `radius`. Must have the following interface:

```
double precision function myImplementationRate(self,node,radius)
 class           (starFormationRateSurfaceDensityDisksClass), intent(inout) :: self
 type            (treeNode                                 ), intent(inout) :: node
 double precision                                           , intent(in   ) :: radius
end double precision function myImplementationRate
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(starFormationRateSurfaceDensityDisksClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(starFormationRateSurfaceDensityDisksClass), intent(inout) :: self
 class(starFormationRateSurfaceDensityDisksClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`intervals` Return a set of integration intervals to use when integrating over the surface density of star formation rate. A default implementation exists. If overridden the following interface must be used:

```
double precision, allocatable, dimension(:,:) function myImplementationIntervals(self,&
&  node,radiusInner, radiusOuter)
 class           (starFormationRateSurfaceDensityDisksClass), intent(inout)         :: &
&  self
 type            (treeNode                                 ), intent(inout), target :: &
&  node
 double precision                                           , intent(in   )         :: &
&  radiusInner, radiusOuter
 end double precision, allocatable, dimension(:,:) function myImplementationIntervals
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (starFormationRateSurfaceDensityDisksClass), intent(inout) :: self
 integer                                           , intent(in   ) :: stateFile
 type   (fgsl_file                                 ), intent(in   ) :: fgslStateFile
 integer(c_size_t                                  ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`starFormationRateSurfaceDensityDisksExtendedSchmidt` The extended Schmidt star formation rate surface density law of Shi et al. [2011] for galactic disks.

`starFormationRateSurfaceDensityDisksBlitz2006` The Blitz and Rosolowsky [2006] star formation rate surface density law for galactic disks.

`starFormationRateSurfaceDensityDisksKrumholz2009` The Krumholz et al. [2009] star formation rate surface density law for galactic disks.

`starFormationRateSurfaceDensityDisksKennicuttSchmidt` A Kennicutt-Schmidt star formation rate surface density for galactic disks.

Timescales for star formation in disks

Additional implementations for timescales for star formation in disks are added using the `starFormationTimescaleDisks` class. The implementation should be placed in a file containing the directive:

```
!# <starFormationTimescaleDisks name="starFormationTimescaleDisksMyImplementation">
!# <description>A short description of the implementation.</description>
!# </starFormationTimescaleDisks>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `starFormationTimescaleDisksMyImplementation`. The file *must* define a type that extends the `starFormationTimescaleDisksClass` class (or extends another type which is itself an extension of the `starFormationTimescaleDisksClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (starFormationTimescaleDisksClass), intent(inout)           :: self
  logical                                  , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (starFormationTimescaleDisksClass), intent(inout)           :: self
 type   (inputParameters                 ), intent(inout)           :: descriptor
 logical                                  , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(starFormationTimescaleDisksClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (starFormationTimescaleDisksClass), intent(inout) :: self
 integer                                  , intent(in   ) :: stateFile
 type   (fgsl_file                        ), intent(in   ) :: fgslStateFile
 integer(c_size_t                         ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (starFormationTimescaleDisksClass)                          , intent(inout) &
&  :: self
 type     (varying_string                 ), allocatable, dimension(:), intent(inout) &
&  :: allowedParameters
 character(len=*                          )                          , intent(in   ) &
&  :: sourceName
 end subroutine myImplementationAllowedParameters
```

`timescale` Returns the timescale (in Gyr) for star formation in the disk component of `node`. Must have the following interface:

```
double precision function myImplementationTimescale(self,node)
 class(starFormationTimescaleDisksClass), intent(inout)         :: self
 type (treeNode                        ), intent(inout), target :: node
end double precision function myImplementationTimescale
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(starFormationTimescaleDisksClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(starFormationTimescaleDisksClass), intent(inout) :: self
 class(starFormationTimescaleDisksClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (starFormationTimescaleDisksClass), intent(inout) :: self
 integer                                  , intent(in   ) :: stateFile
 type   (fgsl_file                        ), intent(in   ) :: fgslStateFile
 integer(c_size_t                         ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`starFormationTimescaleDisksBaugh2005` The Baugh et al. [2005] timescale for star formation in galactic disks.

`starFormationTimescaleDisksIntgrtdSurfaceDensity` A timescale for star formation in galactic disks which computes the timescale by integrating a star formation rate over the disk.

`starFormationTimescaleDisksHaloScaling` A haloScaling timescale for star formation feedback in galactic disks.

`starFormationTimescaleDisksFixed` A fixed timescale for star formation in galactic disks.

`starFormationTimescaleDisksVelocityMaxScaling` A velocityMaxScaling timescale for star formation in galactic disks.

`starFormationTimescaleDisksDynamicalTime` A timescale for star formation in galactic disks which scales with the disk dynamical time.

Timescales for star formation in spheroids

Additional implementations for timescales for star formation in spheroids are added using the `starFormationTimescaleSpher` class. The implementation should be placed in a file containing the directive:

```
!# <starFormationTimescaleSpheroids name="starFormationTimescaleSpheroidsMyImplementation">
!# <description>A short description of the implementation.</description>
!# </starFormationTimescaleSpheroids>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `starFormationTimescaleSpheroidsMyImplementation`. The file *must* define a type that extends the `starFormationTimescaleSpheroidsClass` class (or extends another type which is itself an extension of the `starFormationTimescaleSpheroidsClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (starFormationTimescaleSpheroidsClass), intent(inout)           :: self
  logical                                      , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationDescriptor(self,descriptor,includeMethod)
  class  (starFormationTimescaleSpheroidsClass), intent(inout)           :: self
  type   (inputParameters                     ), intent(inout)           :: descriptor
  logical                                      , intent(in   ), optional :: &
&  includeMethod
 end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAutoHook(self)
  class(starFormationTimescaleSpheroidsClass), intent(inout) :: self
 end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
  class  (starFormationTimescaleSpheroidsClass), intent(inout) :: self
  integer                                      , intent(in   ) :: stateFile
  type   (fgsl_file                           ), intent(in   ) :: fgslStateFile
  integer(c_size_t                            ), intent(in   ) :: stateOperationID
 end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
  class    (starFormationTimescaleSpheroidsClass)                          , intent(&
&  inout) :: self
  type     (varying_string                      ), allocatable, dimension(:), intent(&
&  inout) :: allowedParameters
  character(len=*                               )                          , intent(in &
&    ) :: sourceName
 end subroutine myImplementationAllowedParameters
```

`timescale` Returns the timescale (in Gyr) for star formation in the spheroid component of `node`. Must have the following interface:

```
double precision function myImplementationTimescale(self,node)
 class(starFormationTimescaleSpheroidsClass), intent(inout)         :: self
 type (treeNode                            ), intent(inout), target :: node
end double precision function myImplementationTimescale
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(starFormationTimescaleSpheroidsClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(starFormationTimescaleSpheroidsClass), intent(inout) :: self
 class(starFormationTimescaleSpheroidsClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (starFormationTimescaleSpheroidsClass), intent(inout) :: self
 integer                                      , intent(in   ) :: stateFile
 type   (fgsl_file                            ), intent(in   ) :: fgslStateFile
 integer(c_size_t                             ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`starFormationTimescaleSpheroidsVelocityMaxScaling` A velocityMaxScaling timescale for star formation in galactic spheroids.

`starFormationTimescaleSpheroidsDynamicalTime` A timescale for star formation in galactic spheroids which scales with the spheroid dynamical time.

Stellar Astrophysics

Additional implementations for stellar astrophysics are added using the `stellarAstrophysics` class. The implementation should be placed in a file containing the directive:

```
!# <stellarAstrophysics name="stellarAstrophysicsMyImplementation">
!# <description>A short description of the implementation.</description>
!# </stellarAstrophysics>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `stellarAstrophysics-MyImplementation`. The file *must* define a type that extends the `stellarAstrophysicsClass` class (or extends another type which is itself an extension of the `stellarAstrophysicsClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`massEjected` Returns the mass ejected by a star of given `massInitial` and `metallicity`. Must have the following interface:

```
double precision function myImplementationMassEjected(self,massInitial,metallicity)
 class           (stellarAstrophysicsClass), intent(inout) :: self
 double precision                          , intent(in   ) :: massInitial, metallicity
end double precision function myImplementationMassEjected
```

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
 class  (stellarAstrophysicsClass), intent(inout)           :: self
 logical                          , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (stellarAstrophysicsClass), intent(inout)           :: self
 type   (inputParameters         ), intent(inout)           :: descriptor
 logical                          , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(stellarAstrophysicsClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (stellarAstrophysicsClass), intent(inout) :: self
 integer                          , intent(in   ) :: stateFile
 type   (fgsl_file               ), intent(in   ) :: fgslStateFile
 integer(c_size_t                ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`lifetime` Returns the lifetime of a star of given `massInitial` and `metallicity`. Must have the following interface:

```
double precision function myImplementationLifetime(self,massInitial, metallicity)
 class           (stellarAstrophysicsClass), intent(inout) :: self
 double precision                          , intent(in   ) :: massInitial, metallicity
end double precision function myImplementationLifetime
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
  class    (stellarAstrophysicsClass)                          , intent(inout) :: self
  type     (varying_string         ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
  character(len=*                  )                          , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`massYield` Returns the metal mass yielded by a star of given `massInitial` and `metallicity`. Must have the following interface:

```
 double precision function myImplementationMassYield(self,massInitial, metallicity,&
&  atomIndex)
  class           (stellarAstrophysicsClass), intent(inout)           :: self
  double precision                          , intent(in   )           :: massInitial, &
&  metallicity
  integer                                   , intent(in   ), optional :: atomIndex
 end double precision function myImplementationMassYield
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(stellarAstrophysicsClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`massInitial` Returns the initial mass of a star of given `lifetime` and `metallicity`. Must have the following interface:

```
double precision function myImplementationMassInitial(self,lifetime, metallicity)
 class           (stellarAstrophysicsClass), intent(inout) :: self
 double precision                          , intent(in   ) :: lifetime, metallicity
end double precision function myImplementationMassInitial
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(stellarAstrophysicsClass), intent(inout) :: self
 class(stellarAstrophysicsClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (stellarAstrophysicsClass), intent(inout) :: self
 integer                          , intent(in   ) :: stateFile
 type   (fgsl_file               ), intent(in   ) :: fgslStateFile
 integer(c_size_t                ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`stellarAstrophysicsFile` A stellar astrophysics class in which the stellar properties are read from file and interpolated.

Stellar Feedback

Additional implementations for stellar feedback are added using the `stellarFeedback` class. The implementation should be placed in a file containing the directive:

```
!# <stellarFeedback name="stellarFeedbackMyImplementation">
!# <description>A short description of the implementation.</description>
!# </stellarFeedback>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `stellarFeedbackMyImplementation`. The file *must* define a type that extends the `stellarFeedbackClass` class (or extends another type which is itself an extension of the `stellarFeedbackClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
 class  (stellarFeedbackClass), intent(inout)           :: self
 logical                      , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (stellarFeedbackClass), intent(inout)           :: self
 type   (inputParameters     ), intent(inout)           :: descriptor
 logical                      , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(stellarFeedbackClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (stellarFeedbackClass), intent(inout) :: self
 integer                      , intent(in   ) :: stateFile
 type   (fgsl_file           ), intent(in   ) :: fgslStateFile
 integer(c_size_t            ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (stellarFeedbackClass)                          , intent(inout) :: self
 type     (varying_string      ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*               )                          , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`energyInputCumulative` Return the cumulative energy input from a stellar population of the given `initialMass`, `age`, and `metallicity`. Must have the following interface:

```
double precision function myImplementationEnergyInputCumulative(self,initialMass, age, &
&  metallicity)
 class           (stellarFeedbackClass), intent(inout) :: self
 double precision                      , intent(in   ) :: initialMass, age, metallicity
end double precision function myImplementationEnergyInputCumulative
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(stellarFeedbackClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(stellarFeedbackClass), intent(inout) :: self
 class(stellarFeedbackClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (stellarFeedbackClass), intent(inout) :: self
 integer                      , intent(in   ) :: stateFile
 type   (fgsl_file           ), intent(in   ) :: fgslStateFile
 integer(c_size_t            ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`stellarFeedbackStandard` A stellar feedback class which performs a simple calculation of energy feedback from stellar populations.

Stellar Populations

Additional implementations for stellar populations are added using the `stellarPopulation` class. The implementation should be placed in a file containing the directive:

```
!# <stellarPopulation name="stellarPopulationMyImplementation">
!# <description>A short description of the implementation.</description>
!# </stellarPopulation>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `stellarPopulation-MyImplementation`. The file *must* define a type that extends the `stellarPopulationClass` class (or extends another type which is itself an extension of the `stellarPopulationClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`uniqueID` Return the uniqueID corresponding to this population. A default implementation exists. If overridden the following interface must be used:

```
integer(c_size_t) function myImplementationUniqueID(self)
 class(stellarPopulationClass), intent(inout) :: self
end integer(c_size_t) function myImplementationUniqueID
```

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
 class  (stellarPopulationClass), intent(inout)           :: self
 logical                        , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (stellarPopulationClass), intent(inout)           :: self
 type   (inputParameters       ), intent(inout)           :: descriptor
 logical                        , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(stellarPopulationClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`rateRecycling` Return the rate of recycling from this population. Must have the following interface:

```
double precision function myImplementationRateRecycling(self,abundances_,ageMinimum , &
&  ageMaximum)
 class           (stellarPopulationClass), intent(inout) :: self
 type            (abundances            ), intent(in   ) :: abundances_
 double precision                        , intent(in   ) :: ageMinimum, ageMaximum
end double precision function myImplementationRateRecycling
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (stellarPopulationClass), intent(inout) :: self
 integer                        , intent(in   ) :: stateFile
 type   (fgsl_file             ), intent(in   ) :: fgslStateFile
 integer(c_size_t              ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (stellarPopulationClass)                            , intent(inout) :: self
 type     (varying_string        ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                 )                            , intent(in   ) :: &
&  sourceName
end subroutine myImplementationAllowedParameters
```

`rateEnergy` Return the rate of energy input from this population. Must have the following interface:

```
double precision function myImplementationRateEnergy(self,abundances_,ageMinimum , &
&  ageMaximum)
 class           (stellarPopulationClass), intent(inout) :: self
 type            (abundances            ), intent(in   ) :: abundances_
 double precision                        , intent(in   ) :: ageMinimum, ageMaximum
end double precision function myImplementationRateEnergy
```

`spectra` Return at set of stellar spectra for this population. Must have the following interface:

```
class(stellarPopulationSpectraClass) function myImplementationSpectra(self)
 class(stellarPopulationClass), intent(inout) :: self
end class(stellarPopulationSpectraClass) function myImplementationSpectra
```

`recycledFractionInstantaneous` Return the recycled fraction from this population in the instantaneous approximation. Must have the following interface:

```
double precision function myImplementationRecycledFractionInstantaneous(self)
 class(stellarPopulationClass), intent(inout) :: self
end double precision function myImplementationRecycledFractionInstantaneous
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(stellarPopulationClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(stellarPopulationClass), intent(inout) :: self
 class(stellarPopulationClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`rateYield` Return the rate of element yield from this population. Must have the following interface:

```
 double precision function myImplementationRateYield(self,abundances_,ageMinimum  , &
&  ageMaximum,elementIndex)
  class           (stellarPopulationClass), intent(inout)          :: self
  type            (abundances            ), intent(in   )          :: abundances_
  double precision                         , intent(in   )          :: ageMinimum, &
&  ageMaximum
  integer                                  , intent(in   ), optional :: elementIndex
 end double precision function myImplementationRateYield
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (stellarPopulationClass), intent(inout) :: self
 integer                        , intent(in   ) :: stateFile
 type   (fgsl_file             ), intent(in   ) :: fgslStateFile
 integer(c_size_t              ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

`yieldInstantaneous` Return the metal yield from this population in the instantaneous approximation. Must have the following interface:

```
double precision function myImplementationYieldInstantaneous(self)
 class(stellarPopulationClass), intent(inout) :: self
end double precision function myImplementationYieldInstantaneous
```

Existing implementations are:

`stellarPopulationStandard` A standard stellar population class.

Stellar Population Properties

Additional implementations for stellar population properties are added using the `stellarPopulationProperties` class. The implementation should be placed in a file containing the directive:

```
!# <stellarPopulationProperties name="stellarPopulationPropertiesMyImplementation">
!# <description>A short description of the implementation.</description>
!# </stellarPopulationProperties>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `stellarPopulationPropertiesMyImplementation`. The file *must* define a type that extends the `stellarPopulationPropertiesClass` class (or extends another type which is itself an extension of the `stellarPopulationPropertiesClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (stellarPopulationPropertiesClass), intent(inout)          :: self
  logical                                  , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationDescriptor(self,descriptor,includeMethod)
  class  (stellarPopulationPropertiesClass), intent(inout)          :: self
  type   (inputParameters                 ), intent(inout)          :: descriptor
  logical                                  , intent(in   ), optional :: includeMethod
 end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAutoHook(self)
  class(stellarPopulationPropertiesClass), intent(inout) :: self
 end subroutine myImplementationAutoHook
```

`rates` Returns rates of change of stellar population properties. Must have the following interface:

```
 subroutine myImplementationRates(self,rateStarFormation,abundancesFuel,component,node,&
&  history_,rateMassStellar             , rateMassFuel         , rateEnergyInput,&
&  rateAbundancesFuel          , rateAbundancesStellar,rateLuminosityStellar,&
&  computeRateLuminosityStellar)
  class           (stellarPopulationPropertiesClass), intent(inout) :: self
  double precision                                  , intent(in   ) :: rateStarFormation
  type            (abundances                      ), intent(in   ) :: abundancesFuel
  class           (nodeComponent                   ), intent(in   ) :: component
  type            (treeNode                        ), intent(inout) :: node
  type            (history                         ), intent(inout) :: history_
  double precision                                  , intent(  out) :: rateMassStellar, &
&  rateMassFuel, rateEnergyInput
  type            (abundances                      ), intent(inout) :: &
&  rateAbundancesFuel, rateAbundancesStellar
  type            (stellarLuminosities             ), intent(inout) :: &
&  rateLuminosityStellar
  logical                                           , intent(in   ) :: &
&  computeRateLuminosityStellar
 end subroutine myImplementationRates
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
  class  (stellarPopulationPropertiesClass), intent(inout) :: self
  integer                                  , intent(in   ) :: stateFile
```

```
 type   (fgsl_file                      ), intent(in   ) :: fgslStateFile
 integer(c_size_t                       ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`historyCount` Return the number of stellar population property histories which must be stored. Must have the following interface:

```
integer function myImplementationHistoryCount(self)
 class(stellarPopulationPropertiesClass), intent(inout) :: self
end integer function myImplementationHistoryCount
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (stellarPopulationPropertiesClass)                          , intent(inout) &
&  :: self
 type     (varying_string                 ), allocatable, dimension(:), intent(inout) &
&  :: allowedParameters
 character(len=*                          )                          , intent(in   ) &
&  :: sourceName
 end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(stellarPopulationPropertiesClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`historyCreate` Create histories needed to store stellar population properties. Must have the following interface:

```
subroutine myImplementationHistoryCreate(self,node,history_)
 class(stellarPopulationPropertiesClass), intent(inout) :: self
 type (treeNode                        ), intent(inout) :: node
 type (history                         ), intent(inout) :: history_
end subroutine myImplementationHistoryCreate
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(stellarPopulationPropertiesClass), intent(inout) :: self
 class(stellarPopulationPropertiesClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (stellarPopulationPropertiesClass), intent(inout) :: self
 integer                                  , intent(in   ) :: stateFile
 type   (fgsl_file                      ), intent(in   ) :: fgslStateFile
 integer(c_size_t                       ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

`scales` Return scaling factors of stellar population properties for an ODE solver. Must have the following interface:

```
subroutine myImplementationScales(self,massStellar,abundancesStellar,history_)
 class           (stellarPopulationPropertiesClass), intent(inout) :: self
 double precision                                  , intent(in   ) :: massStellar
 type            (abundances                      ), intent(in   ) :: abundancesStellar
 type            (history                         ), intent(inout) :: history_
end subroutine myImplementationScales
```

Existing implementations are:

`stellarPopulationPropertiesNoninstantaneous` A stellar population properties class based on the noninstantaneous recycling approximation.

`stellarPopulationPropertiesInstantaneous` A stellar population properties class based on the instantaneous recycling approximation.

Stellar Population Selectors

Additional implementations for stellar population selectors are added using the `stellarPopulationSelector` class. The implementation should be placed in a file containing the directive:

```
!# <stellarPopulationSelector name="stellarPopulationSelectorMyImplementation">
!# <description>A short description of the implementation.</description>
!# </stellarPopulationSelector>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `stellarPopulationSelectorMyImplementation`. The file *must* define a type that extends the `stellarPopulationSelectorClass` class (or extends another type which is itself an extension of the `stellarPopulationSelectorClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (stellarPopulationSelectorClass), intent(inout)           :: self
  logical                                , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (stellarPopulationSelectorClass), intent(inout)           :: self
 type   (inputParameters               ), intent(inout)           :: descriptor
 logical                                , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(stellarPopulationSelectorClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (stellarPopulationSelectorClass), intent(inout) :: self
 integer                                , intent(in   ) :: stateFile
 type   (fgsl_file                      ), intent(in   ) :: fgslStateFile
 integer(c_size_t                       ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (stellarPopulationSelectorClass)                          , intent(inout) ::&
&  self
 type     (varying_string               ), allocatable, dimension(:), intent(inout) ::&
&  allowedParameters
 character(len=*                         )                          , intent(in   ) ::&
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`isStarFormationRateDependent` Return true if the selection of stellar population is dependent on star formation rate. Must have the following interface:

```
logical function myImplementationIsStarFormationRateDependent(self)
 class(stellarPopulationSelectorClass), intent(inout) :: self
end logical function myImplementationIsStarFormationRateDependent
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(stellarPopulationSelectorClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(stellarPopulationSelectorClass), intent(inout) :: self
 class(stellarPopulationSelectorClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (stellarPopulationSelectorClass), intent(inout) :: self
 integer                                , intent(in   ) :: stateFile
 type   (fgsl_file                      ), intent(in   ) :: fgslStateFile
 integer(c_size_t                       ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

`select` Return a stellar population. Must have the following interface:

```
class(stellarPopulationClass) function myImplementationSelect(self,rateStarFormation,&
&  abundances_,component)
 class           (stellarPopulationSelectorClass), intent(inout) :: self
 double precision                                , intent(in   ) :: rateStarFormation
 type            (abundances                     ), intent(in   ) :: abundances_
 class           (nodeComponent                  ), intent(in   ) :: component
end class(stellarPopulationClass) function myImplementationSelect
```

Existing implementations are:

`stellarPopulationSelectorDiskSpheroid` A stellar population selector class which returns a different population for disks and spheroids.

`stellarPopulationSelectorFixed` A fixed stellar population selector class.

Stellar Population Spectra

Additional implementations for stellar population spectra are added using the `stellarPopulationSpectra` class. The implementation should be placed in a file containing the directive:

```
!# <stellarPopulationSpectra name="stellarPopulationSpectraMyImplementation">
!# <description>A short description of the implementation.</description>
!# </stellarPopulationSpectra>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `stellarPopulationSpectraMyImplementation`. The file *must* define a type that extends the `stellarPopulationSpectraClass` class (or extends another type which is itself an extension of the `stellarPopulationSpectraClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
 class  (stellarPopulationSpectraClass), intent(inout)          :: self
 logical                               , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (stellarPopulationSpectraClass), intent(inout)          :: self
 type   (inputParameters              ), intent(inout)          :: descriptor
 logical                               , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`luminosity` Return the luminosity (in units of $L_\odot$ Hz^{-1}) for a stellar population, composition `abundances`, of the given `age` (in Gyr), at the specified `wavelength` (in Angstroms). Must have the following interface:

```
 double precision function myImplementationLuminosity(self,abundancesStellar,age        &
&        , wavelength,status)
  class           (stellarPopulationSpectraClass), intent(inout)           :: self
  type            (abundances                   ), intent(in   )           :: &
&  abundancesStellar
  double precision                                , intent(in   )           :: age, &
&  wavelength
  integer                                         , intent(  out), optional :: status
 end double precision function myImplementationLuminosity
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(stellarPopulationSpectraClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (stellarPopulationSpectraClass), intent(inout) :: self
 integer                               , intent(in   ) :: stateFile
 type   (fgsl_file                     ), intent(in   ) :: fgslStateFile
 integer(c_size_t                      ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`tabulation` Return a tabulation of ages and metallicities at which stellar spectra should be tabulated. Must have the following interface:

```
 subroutine myImplementationTabulation(self,agesCount, metallicitiesCount,ages     , &
&  metallicity)
  class           (stellarPopulationSpectraClass)                          , intent(&
&  inout) :: self
  integer                                                                  , intent(  &
&  out) :: agesCount, metallicitiesCount
  double precision                                , allocatable, dimension(:), intent(  &
&  out) :: ages, metallicity
 end subroutine myImplementationTabulation
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
  class    (stellarPopulationSpectraClass)                          , intent(inout) :: &
&  self
  type     (varying_string              ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
  character(len=*                       )                          , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`wavelengths` Return a tabulation of wavelengths at which stellar spectra are defined. Must have the following interface:

```
 subroutine myImplementationWavelengths(self,wavelengthsCount,wavelengths)
  class           (stellarPopulationSpectraClass)                          , intent(&
&  inout) :: self
  integer                                                                  , intent(  &
&  out) :: wavelengthsCount
  double precision                                , allocatable, dimension(:), intent(  &
&  out) :: wavelengths
 end subroutine myImplementationWavelengths
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(stellarPopulationSpectraClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(stellarPopulationSpectraClass), intent(inout) :: self
 class(stellarPopulationSpectraClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`wavelengthInterval` At a given wavelength, return the wavelength interval in the tabulation of wavelength for which stellar spectra are defined. Must have the following interface:

```
double precision function myImplementationWavelengthInterval(self,wavelength)
 class           (stellarPopulationSpectraClass), intent(inout) :: self
 double precision                               , intent(in   ) :: wavelength
end double precision function myImplementationWavelengthInterval
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (stellarPopulationSpectraClass), intent(inout) :: self
 integer                               , intent(in   ) :: stateFile
 type   (fgsl_file                     ), intent(in   ) :: fgslStateFile
 integer(c_size_t                      ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`stellarPopulationSpectraFile` Provides stellar population spectra via interpolation in a tabulation read from file. This should be an HDF5 file with the following structure:

```
ages                    Dataset {ageCount}
metallicities           Dataset {metallicityCount}
spectra                 Dataset {metallicityCount, ageCount, metallicityCount}
wavelengths             Dataset {wavelengthCount}
```

where the datasets contain the tabulated ages (in Gyr), metallicities (logarithmic, relative to Solar), wavelengths (in Å) and spectra (in $L_\odot$ Hz^{-1}). Currently, the following pre-computed stellar spectra files are available as a separate download from http://users.obs.carnegiescience.edu/abenson/galacticus/data/SSP_Data.tar.bz2:

`stellarPopulations/SSP_Spectra_Conroy-et-al_v2.0_imfSalpeter.hdf5` Corresponds to a Salpeter IMF computed using v2.0 of the FSPS code;

`stellarPopulations/SSP_Spectra_Conroy-et-al_v2.1_imfSalpeter.hdf5` Corresponds to a Salpeter IMF computed using v2.1 of the FSPS code;

`stellarPopulations/SSP_Spectra_Conroy-et-al_v2.1_imfChabrier.hdf5` Corresponds to a Chabrier IMF computed using v2.1 of the FSPS code;

`stellarPopulations/SSP_Spectra_Conroy-et-al_v2.2_imfChabrier.hdf5` Corresponds to a Chabrier IMF computed using v2.2 of the FSPS code;

`stellarPopulations/SSP_Spectra_Conroy-et-al_v2.2_imfKennicutt.hdf5` Corresponds to a Kennicutt IMF computed using v2.2 of the FSPS code;

`stellarPopulations/SSP_Spectra_Conroy-et-al_v2.2_imfBaugh2005TopHeavy.hdf5` Corresponds to the top-heavy IMF of Baugh et al. [2005] computed using v2.2 of the FSPS code;

`stellarPopulations/SSP_Spectra_Maraston_hbMorphologyRed_imfKroupa.hdf5` The spectra from Maraston [2005] for a Kroupa IMF and a red horizontal branch morphology;

`stellarPopulations/SSP_Spectra_Maraston_hbMorphologyRed_imfSalpeter.hdf5` The spectra from Maraston [2005] for a Salpeter IMF and a red horizontal branch morphology;

`stellarPopulations/SSP_Spectra_BC2003_highResolution_imfChabrier.hdf5` The (high resolution) spectra from Bruzual and Charlot [2003] for a Chabrier IMF, using Padova 1994 tracks;

`stellarPopulations/SSP_Spectra_BC2003_highResolution_imfSalpeter.hdf5` The (high resolution) spectra from Bruzual and Charlot [2003] for a Salpeter IMF, using Padova 1994 tracks;

`stellarPopulations/SSP_Spectra_BC2003_lowResolution_imfChabrier.hdf5` The (low resolution) spectra from Bruzual and Charlot [2003] for a Chabrier IMF, using Padova 1994 tracks;

`stellarPopulations/SSP_Spectra_BC2003_lowResolution_imfSalpeter.hdf5` The (low resolution) spectra from Bruzual and Charlot [2003] for a Salpeter IMF, using Padova 1994 tracks;

`stellarPopulations/SSP_Spectra_Grasil_gkn15rd_ken.hdf5` The spectra used by GRASIL for a Kennicutt IMF;

`stellarPopulations/SSP_Spectra_Grasil_gkn1rd_ken.hdf5` The spectra used by GRASIL for a Kennicutt IMF;

`stellarPopulations/SSP_Spectra_Grasil_gsrdk0b_sal.hdf5` The spectra used by GRASIL for a Salpeter IMF;

`stellarPopulations/SSP_Spectra_Grasil_imf27_kro.hdf5` Spectra used by GRASIL.

Note that the high resolution spectra from Bruzual and Charlot [2003] may require you to adjust the `[stellarPopulationLuminosityIntegrationToleranceRelative]` parameter to a larger value[7]. The sharp features in these high resolution spectra can be difficult to integrate. Scripts to convert the data provided by Maraston [2005] and Bruzual and Charlot [2003] into GALACTICUS's format are provided in the `scripts/ssps` folder.

`stellarPopulationSpectraFSPS` Provides stellar population spectra utilizing the FSPS package [Conroy et al., 2009]. If necessary, the `FSPS` code will be downloaded, patched and compiled and run to generate spectra. These tabulations are then stored to file for later re-use.

Postprocessors for stellar population spectra

Additional implementations for postprocessors for stellar population spectra are added using the `stellarPopulationSpectraP` class. The implementation should be placed in a file containing the directive:

```
!# <stellarPopulationSpectraPostprocessor name="stellarPopulationSpectraPostprocessorMyImplementation">
!# <description>A short description of the implementation.</description>
!# </stellarPopulationSpectraPostprocessor>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `stellarPopulationSpectraPostprocessorMyImplementation`. The file *must* define a type that extends the `stellarPopulationSpectraPostprocessorClass` class (or extends another type which is itself an extension of the `stellarPopulationSpectraPostprocessorClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (stellarPopulationSpectraPostprocessorClass), intent(inout)          :: self
  logical                                            , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationDescriptor(self,descriptor,includeMethod)
  class  (stellarPopulationSpectraPostprocessorClass), intent(inout)          :: self
  type   (inputParameters                           ), intent(inout)          :: &
&  descriptor
  logical                                            , intent(in   ), optional :: &
&  includeMethod
 end subroutine myImplementationDescriptor
```

[7] Or, alternatively, set `[stellarPopulationLuminosityIntegrationToleranceDegrade]=true`. This will cause GALACTICUS to increase the tolerance as necessary to get the integrals to converge—issuing warnings each time the tolerance is increased.

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(stellarPopulationSpectraPostprocessorClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (stellarPopulationSpectraPostprocessorClass), intent(inout) :: self
 integer                                            , intent(in   ) :: stateFile
 type   (fgsl_file                                  ), intent(in   ) :: fgslStateFile
 integer(c_size_t                                   ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (stellarPopulationSpectraPostprocessorClass)                          , &
&  intent(inout) :: self
 type     (varying_string                            ), allocatable, dimension(:), &
&  intent(inout) :: allowedParameters
 character(len=*                                     )                          , &
&  intent(in   ) :: sourceName
 end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(stellarPopulationSpectraPostprocessorClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(stellarPopulationSpectraPostprocessorClass), intent(inout) :: self
 class(stellarPopulationSpectraPostprocessorClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (stellarPopulationSpectraPostprocessorClass), intent(inout) :: self
 integer                                            , intent(in   ) :: stateFile
 type   (fgsl_file                                  ), intent(in   ) :: fgslStateFile
 integer(c_size_t                                   ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

`multiplier` Return the multiplicative modification to the spectrum. Must have the following interface:

```
double precision function myImplementationMultiplier(self,wavelength, age, redshift)
 class           (stellarPopulationSpectraPostprocessorClass), intent(inout) :: self
 double precision                                            , intent(in   ) :: &
&  wavelength, age, redshift
 end double precision function myImplementationMultiplier
```

Existing implementations are:

`stellarPopulationSpectraPostprocessorMeiksin2006` Multiplier the Meiksin [2006] calculation of the attenuation of spectra by the intergalactic medium.

`stellarPopulationSpectraPostprocessorInoue2014` Apply the Inoue et al. [2014] calculation of the attenuation of spectra by the intergalactic medium.

`stellarPopulationSpectraPostprocessorUnescaped` Retains only unescaped stellar populations.

`stellarPopulationSpectraPostprocessorIdentity` Performs an indentity postprocessing of spectra.

`stellarPopulationSpectraPostprocessorMadau1995` Multiplier the Madau [1995] calculation of the attenuation of spectra by the intergalactic medium.

`stellarPopulationSpectraPostprocessorRecent` Retains only recent stellar populations.

`stellarPopulationSpectraPostprocessorLycSuppress` A stellar population spectrum postprocessor which completely supresses the Lyman continuum.

`stellarPopulationSpectraPostprocessorSequence` A sequence stellar population spectra postprocessor class.

Builder for postprocessors for stellar population spectra

Additional implementations for builder for postprocessors for stellar population spectra are added using the `stellarPopulationSpectraPostprocessorBuilder` class. The implementation should be placed in a file containing the directive:

```
!# <stellarPopulationSpectraPostprocessorBuilder name="stellarPopulationSpectraPostprocessorBuilderMyImp
!# <description>A short description of the implementation.</description>
!# </stellarPopulationSpectraPostprocessorBuilder>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `stellarPopulationSpectraPostprocessorBuilderMyImplementation`. The file *must* define a type that extends the `stellarPopulationSpectraPostprocessorBuilderClass` class (or extends another type which is itself an extension of the `stellarPopulationSpectraPostprocessorBuilderClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (stellarPopulationSpectraPostprocessorBuilderClass), intent(inout)          ::&
&   self
  logical                                                   , intent(in   ), optional ::&
&   includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationDescriptor(self,descriptor,includeMethod)
  class  (stellarPopulationSpectraPostprocessorBuilderClass), intent(inout)          ::&
&   self
  type   (inputParameters                                  ), intent(inout)          ::&
&   descriptor
  logical                                                   , intent(in   ), optional ::&
&   includeMethod
 end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAutoHook(self)
  class(stellarPopulationSpectraPostprocessorBuilderClass), intent(inout) :: self
 end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
  class  (stellarPopulationSpectraPostprocessorBuilderClass), intent(inout) :: self
  integer                                                   , intent(in   ) :: stateFile
  type   (fgsl_file                                        ), intent(in   ) :: &
&  fgslStateFile
  integer(c_size_t                                         ), intent(in   ) :: &
&  stateOperationID
 end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
  class    (stellarPopulationSpectraPostprocessorBuilderClass)                       &
&   , intent(inout) :: self
  type     (varying_string                                    ), allocatable, dimension&
&  (:), intent(inout) :: allowedParameters
  character(len=*                                             )                       &
&   , intent(in   ) :: sourceName
 end subroutine myImplementationAllowedParameters
```

`build` Build and return a postprocessor. Must have the following interface:

```
 class(stellarPopulationSpectraPostprocessorClass) function myImplementationBuild(self,&
&  descriptor)
  class(stellarPopulationSpectraPostprocessorBuilderClass), intent(inout) :: self
  type (varying_string                                   ), intent(in   ) :: descriptor
 end class(stellarPopulationSpectraPostprocessorClass) function myImplementationBuild
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(stellarPopulationSpectraPostprocessorBuilderClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(stellarPopulationSpectraPostprocessorBuilderClass), intent(inout) :: self
 class(stellarPopulationSpectraPostprocessorBuilderClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (stellarPopulationSpectraPostprocessorBuilderClass), intent(inout) :: self
 integer                                                   , intent(in   ) :: stateFile
 type   (fgsl_file                                        ), intent(in   ) :: &
&  fgslStateFile
 integer(c_size_t                                         ), intent(in   ) :: &
&  stateOperationID
 end subroutine myImplementationStateRestore
```

Existing implementations are:

`stellarPopulationSpectraPostprocessorBuilderLookup` A stellar population spectra postprocessor builder which simply looks up postprocessors by name.

Stellar Spectra Dust Attenuation

Additional implementations for stellar spectra dust attenuation are added using the `stellarSpectraDustAttenuation` class. The implementation should be placed in a file containing the directive:

```
!# <stellarSpectraDustAttenuation name="stellarSpectraDustAttenuationMyImplementation">
!# <description>A short description of the implementation.</description>
!# </stellarSpectraDustAttenuation>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `stellarSpectraDustAttenuationMyImplementation`. The file *must* define a type that extends the `stellarSpectraDustAttenuationClass` class (or extends another type which is itself an extension of the `stellarSpectraDustAttenuationClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`isSeparable` Return true if the attenuation is separable into a product of functions of wavelength, age, and V-band attenuation. A default implementation exists. If overridden the following interface must be used:

```
logical function myImplementationIsSeparable(self)
 class(stellarSpectraDustAttenuationClass), intent(inout) :: self
end logical function myImplementationIsSeparable
```

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (stellarSpectraDustAttenuationClass), intent(inout)           :: self
  logical                                    , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (stellarSpectraDustAttenuationClass), intent(inout)           :: self
 type   (inputParameters                   ), intent(inout)           :: descriptor
 logical                                    , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(stellarSpectraDustAttenuationClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (stellarSpectraDustAttenuationClass), intent(inout) :: self
 integer                                    , intent(in   ) :: stateFile
 type   (fgsl_file                         ), intent(in   ) :: fgslStateFile
 integer(c_size_t                          ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (stellarSpectraDustAttenuationClass)                          , intent(inout&
&  ) :: self
 type     (varying_string                    ), allocatable, dimension(:), intent(inout&
&  ) :: allowedParameters
 character(len=*                             )                          , intent(in   &
&  ) :: sourceName
 end subroutine myImplementationAllowedParameters
```

`attenuation` Return the attenuation, in magnitudes, of stellar spectra due to dust at the given wavelength, age, and V-band extinction. Must have the following interface:

```
 double precision function myImplementationAttenuation(self,wavelength, age, &
&  vBandAttenuation)
  class           (stellarSpectraDustAttenuationClass), intent(inout) :: self
  double precision                                    , intent(in   ) :: wavelength, age&
&  , vBandAttenuation
 end double precision function myImplementationAttenuation
```

`isAgeDependent` Return true if the attenuation may depend on the age of the stellar population. A default implementation exists. If overridden the following interface must be used:

```
logical function myImplementationIsAgeDependent(self)
 class(stellarSpectraDustAttenuationClass), intent(inout) :: self
end logical function myImplementationIsAgeDependent
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(stellarSpectraDustAttenuationClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(stellarSpectraDustAttenuationClass), intent(inout) :: self
 class(stellarSpectraDustAttenuationClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (stellarSpectraDustAttenuationClass), intent(inout) :: self
 integer                                    , intent(in   ) :: stateFile
 type   (fgsl_file                          ), intent(in   ) :: fgslStateFile
 integer(c_size_t                           ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`stellarSpectraDustAttenuationGordon2003` Returns the dust attenuation of stellar spectra according to the model of Gordon et al. [2003].

`stellarSpectraDustAttenuationTabulated` Returns the dust attenuation of stellar spectra from a tabulated relation.

`stellarSpectraDustAttenuationCharlotFall2000` Returns the dust attenuation of stellar spectra according to the model of Charlot and Fall [2000].

`stellarSpectraDustAttenuationWittGordon2000` Returns the dust attenuation of stellar spectra according to the model of Witt and Gordon [2000].

`stellarSpectraDustAttenuationZero` Returns a zero dust attenuation.

`stellarSpectraDustAttenuationCardelli1989` Returns the dust attenuation of stellar spectra according to the model of Cardelli et al. [1989].

`stellarSpectraDustAttenuationCalzetti2000` Returns the dust attenuation of stellar spectra according to the model of Calzetti et al. [2000].

`stellarSpectraDustAttenuationPrevotBouchet` Returns the dust attenuation of stellar spectra according to the model of Prevot et al. [1984] and Bouchet et al. [1985].

Stellar Tracks

Additional implementations for stellar tracks are added using the `stellarTracks` class. The implementation should be placed in a file containing the directive:

```
!# <stellarTracks name="stellarTracksMyImplementation">
!# <description>A short description of the implementation.</description>
!# </stellarTracks>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `stellarTracksMyImplementation`. The file *must* define a type that extends the `stellarTracksClass` class (or extends another type which is itself an extension of the `stellarTracksClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`temperatureEffective` Returns the effective temperature of a star of given `initialMass`, `metallicity` and `age`. Must have the following interface:

```
 double precision function myImplementationTemperatureEffective(self,initialMass, &
&  metallicity, age)
  class           (stellarTracksClass), intent(inout) :: self
  double precision                    , intent(in   ) :: initialMass, metallicity, age
 end double precision function myImplementationTemperatureEffective
```

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (stellarTracksClass), intent(inout)           :: self
  logical                    , intent(in   ), optional :: includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationDescriptor(self,descriptor,includeMethod)
  class  (stellarTracksClass), intent(inout)           :: self
  type   (inputParameters   ), intent(inout)           :: descriptor
  logical                    , intent(in   ), optional :: includeMethod
 end subroutine myImplementationDescriptor
```

`luminosity` Returns the bolometric luminosity of a star of given `initialMass`, `metallicity` and `age`. Must have the following interface:

```
double precision function myImplementationLuminosity(self,initialMass, metallicity, age&
&  )
 class           (stellarTracksClass), intent(inout) :: self
 double precision                    , intent(in   ) :: initialMass, metallicity, age
end double precision function myImplementationLuminosity
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(stellarTracksClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (stellarTracksClass), intent(inout) :: self
 integer                    , intent(in   ) :: stateFile
 type   (fgsl_file         ), intent(in   ) :: fgslStateFile
 integer(c_size_t          ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (stellarTracksClass)                          , intent(inout) :: self
 type     (varying_string    ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*             )                          , intent(in   ) :: sourceName
end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(stellarTracksClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(stellarTracksClass), intent(inout) :: self
 class(stellarTracksClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (stellarTracksClass), intent(inout) :: self
 integer                    , intent(in   ) :: stateFile
 type   (fgsl_file         ), intent(in   ) :: fgslStateFile
 integer(c_size_t          ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`stellarTracksFile` A stellar tracks class in which the tracks are read from file and interpolated.

Stellar Winds

Additional implementations for stellar winds are added using the `stellarWinds` class. The implementation should be placed in a file containing the directive:

```
!# <stellarWinds name="stellarWindsMyImplementation">
!# <description>A short description of the implementation.</description>
!# </stellarWinds>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `stellarWindsMyImplementation`. The file *must* define a type that extends the `stellarWindsClass` class (or extends another type which is itself an extension of the `stellarWindsClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`velocityTerminal` Return the terminal velocity (in km/s) of winds from stars of given `initialMass`, `age` and `metallicity`. Must have the following interface:

```
 double precision function myImplementationVelocityTerminal(self,initialMass, age, &
&  metallicity)
  class           (stellarWindsClass), intent(inout) :: self
  double precision                   , intent(in   ) :: initialMass, age, metallicity
 end double precision function myImplementationVelocityTerminal
```

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (stellarWindsClass), intent(inout)           :: self
  logical                   , intent(in   ), optional :: includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationDescriptor(self,descriptor,includeMethod)
  class  (stellarWindsClass), intent(inout)           :: self
  type   (inputParameters  ), intent(inout)           :: descriptor
  logical                   , intent(in   ), optional :: includeMethod
 end subroutine myImplementationDescriptor
```

`rateMassLoss` Return the mass loss rate (in $M_\odot$/Gyr) from stars of given `initialMass`, `age` and `metallicity`. Must have the following interface:

```
double precision function myImplementationRateMassLoss(self,initialMass, age, &
&  metallicity)
 class           (stellarWindsClass), intent(inout) :: self
 double precision                   , intent(in   ) :: initialMass, age, metallicity
end double precision function myImplementationRateMassLoss
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(stellarWindsClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (stellarWindsClass), intent(inout) :: self
 integer                   , intent(in   ) :: stateFile
 type   (fgsl_file        ), intent(in   ) :: fgslStateFile
 integer(c_size_t         ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (stellarWindsClass)                          , intent(inout) :: self
 type     (varying_string   ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*            )                          , intent(in   ) :: sourceName
end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(stellarWindsClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(stellarWindsClass), intent(inout) :: self
 class(stellarWindsClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (stellarWindsClass), intent(inout) :: self
 integer                   , intent(in   ) :: stateFile
 type   (fgsl_file        ), intent(in   ) :: fgslStateFile
 integer(c_size_t         ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`stellarWindsLeitherer1992` A stellar winds class based on Leitherer et al. [1992].

Supernovae Type Ia

Additional implementations for supernovae type ia are added using the `supernovaePopulationIII` class. The implementation should be placed in a file containing the directive:

```
!# <supernovaePopulationIII name="supernovaePopulationIIIMyImplementation">
!# <description>A short description of the implementation.</description>
!# </supernovaePopulationIII>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `supernovaePopulationIIIMyImplementation`. The file *must* define a type that extends the `supernovaePopulationIIIClass` class (or extends another type which is itself an extension of the `supernovaePopulationIIIClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
 class  (supernovaePopulationIIIClass), intent(inout)           :: self
 logical                              , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (supernovaePopulationIIIClass), intent(inout)           :: self
 type   (inputParameters             ), intent(inout)           :: descriptor
 logical                              , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(supernovaePopulationIIIClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (supernovaePopulationIIIClass), intent(inout) :: self
 integer                              , intent(in   ) :: stateFile
 type   (fgsl_file                   ), intent(in   ) :: fgslStateFile
 integer(c_size_t                    ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (supernovaePopulationIIIClass)                          , intent(inout) :: &
&  self
 type     (varying_string             ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                      )                          , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(supernovaePopulationIIIClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(supernovaePopulationIIIClass), intent(inout) :: self
 class(supernovaePopulationIIIClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (supernovaePopulationIIIClass), intent(inout) :: self
 integer                              , intent(in   ) :: stateFile
 type   (fgsl_file                   ), intent(in   ) :: fgslStateFile
 integer(c_size_t                    ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

`energyCumulative` Return the cumulative energy input from Population III supernovae from stars of given `initialMass`, `age` and `metallicity`. Must have the following interface:

```
double precision function myImplementationEnergyCumulative(self,initialMass, age, &
&  metallicity)
 class           (supernovaePopulationIIIClass), intent(inout) :: self
 double precision                              , intent(in   ) :: initialMass, age, &
&  metallicity
 end double precision function myImplementationEnergyCumulative
```

Existing implementations are:

`supernovaePopulationIIIHegerWoosley2002` A Population III supernovae class based on Heger and Woosley [2002].

Supernovae Type Ia

Additional implementations for supernovae type ia are added using the `supernovaeTypeIa` class. The implementation should be placed in a file containing the directive:

```
!# <supernovaeTypeIa name="supernovaeTypeIaMyImplementation">
!# <description>A short description of the implementation.</description>
!# </supernovaeTypeIa>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `supernovaeTypeIaMyImplementation`. The file *must* define a type that extends the `supernovaeTypeIaClass` class (or extends another type which is itself an extension of the `supernovaeTypeIaClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (supernovaeTypeIaClass), intent(inout)           :: self
  logical                       , intent(in   ), optional :: includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationDescriptor(self,descriptor,includeMethod)
  class  (supernovaeTypeIaClass), intent(inout)           :: self
  type   (inputParameters      ), intent(inout)           :: descriptor
  logical                       , intent(in   ), optional :: includeMethod
 end subroutine myImplementationDescriptor
```

`number` Return the cumulative number of Type Ia supernovae from a stellar population of the given `initialMass`, `age`, and `metallicity`. Must have the following interface:

```
 double precision function myImplementationNumber(self,initialMass, age, metallicity)
  class           (supernovaeTypeIaClass), intent(inout) :: self
  double precision                       , intent(in   ) :: initialMass, age, &
&  metallicity
 end double precision function myImplementationNumber
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(supernovaeTypeIaClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (supernovaeTypeIaClass), intent(inout) :: self
 integer                       , intent(in   ) :: stateFile
 type   (fgsl_file            ), intent(in   ) :: fgslStateFile
 integer(c_size_t             ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (supernovaeTypeIaClass)                            , intent(inout) :: self
 type     (varying_string       ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                )                            , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`yield` Return the cumulative yield from Type Ia supernoave from a stellar population of the given `initialMass`, `age`, and `metallicity`. Must have the following interface:

```
double precision function myImplementationYield(self,initialMass, age, metallicity,&
&  atomIndex)
 class           (supernovaeTypeIaClass), intent(inout)           :: self
 double precision                       , intent(in   )           :: initialMass, age, &
&  metallicity
 integer                                , intent(in   ), optional :: atomIndex
end double precision function myImplementationYield
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(supernovaeTypeIaClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(supernovaeTypeIaClass), intent(inout) :: self
 class(supernovaeTypeIaClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (supernovaeTypeIaClass), intent(inout) :: self
 integer                       , intent(in   ) :: stateFile
 type   (fgsl_file            ), intent(in   ) :: fgslStateFile
 integer(c_size_t             ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`supernovaeTypeIaNagashima2005` A supernovae type Ia class based on Nagashima et al. [2005].

Survey Geometry

Additional implementations for survey geometry are added using the `surveyGeometry` class. The implementation should be placed in a file containing the directive:

```
!# <surveyGeometry name="surveyGeometryMyImplementation">
!# <description>A short description of the implementation.</description>
!# </surveyGeometry>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `surveyGeometryMyImplementation`. The file *must* define a type that extends the `surveyGeometryClass` class (or extends another type which is itself an extension of the `surveyGeometryClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`windowFunctionAvailable` Returns true if survey 3-D window functions are available. Must have the following interface:

```
logical function myImplementationWindowFunctionAvailable(self)
 class(surveyGeometryClass), intent(inout) :: self
end logical function myImplementationWindowFunctionAvailable
```

`angularPower` Return C_ℓ^{ij}, where $(2\ell+1)C_\ell^{ij} = \sum_{m=-\ell}^{+\ell} \Psi_{\ell m}^i \Psi_{\ell m}^{j*}$, and $\Psi_{\ell m}^i$ are the cofficients of the spherical harmonic expansion of the $i^{\rm th}$ field. Must have the following interface:

```
double precision function myImplementationAngularPower(self,i,j,l)
 class  (surveyGeometryClass), intent(inout) :: self
 integer                     , intent(in   ) :: i, j, l
end double precision function myImplementationAngularPower
```

`angularPowerMaximumDegree` Return the maximum degree, $\ell_{\rm max}$, for which the angular power is available. A default implementation exists. If overridden the following interface must be used:

```
integer function myImplementationAngularPowerMaximumDegree(self)
 class(surveyGeometryClass), intent(inout) :: self
end integer function myImplementationAngularPowerMaximumDegree
```

`volumeMaximum` Returns the maximum volume (in Mpc^3) at which a galaxy of the specified `mass` (in $M_\odot$) could be detected. A default implementation exists. If overridden the following interface must be used:

```
double precision function myImplementationVolumeMaximum(self,mass,field)
 class           (surveyGeometryClass), intent(inout)           :: self
 double precision                     , intent(in   )           :: mass
 integer                              , intent(in   ), optional :: field
end double precision function myImplementationVolumeMaximum
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(surveyGeometryClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`pointIncluded` Return true if the given Cartesian point lies within the survey bounds for the given mass limit. Must have the following interface:

```
logical function myImplementationPointIncluded(self,point,mass)
 class           (surveyGeometryClass)              , intent(inout) :: self
 double precision                     , dimension(3), intent(in   ) :: point
 double precision                                    , intent(in   ) :: mass
end logical function myImplementationPointIncluded
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (surveyGeometryClass), intent(inout)           :: self
 type   (inputParameters    ), intent(inout)           :: descriptor
 logical                     , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
 class  (surveyGeometryClass), intent(inout)           :: self
 logical                     , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(surveyGeometryClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (surveyGeometryClass), intent(inout) :: self
 integer                     , intent(in   ) :: stateFile
```

```
 type   (fgsl_file          ), intent(in   ) :: fgslStateFile
 integer(c_size_t           ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`fieldCount` Returns the number of distinct fields included in the survey. A default implementation exists. If overridden the following interface must be used:

```
integer function myImplementationFieldCount(self)
 class(surveyGeometryClass), intent(inout) :: self
end integer function myImplementationFieldCount
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (surveyGeometryClass)                          , intent(inout) :: self
 type     (varying_string     ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*              )                          , intent(in   ) :: sourceName
end subroutine myImplementationAllowedParameters
```

`solidAngle` Return the solid angle (in steradians) of the survey. Must have the following interface:

```
double precision function myImplementationSolidAngle(self,field)
 class  (surveyGeometryClass), intent(inout)           :: self
 integer                     , intent(in   ), optional :: field
end double precision function myImplementationSolidAngle
```

`distanceMaximum` Returns the maximum distance (in Mpc) at which a galaxy of the specified `mass` (in $M_\odot$) could be detected. Must have the following interface:

```
 double precision function myImplementationDistanceMaximum(self,mass , magnitudeAbsolute&
&  , luminosity,field)
  class           (surveyGeometryClass), intent(inout)           :: self
  double precision                     , intent(in   ), optional :: mass, &
&  magnitudeAbsolute, luminosity
  integer                              , intent(in   ), optional :: field
 end double precision function myImplementationDistanceMaximum
```

`windowFunctions` Returns the window functions on a grid of the specified size (`gridCount` cells in each dimension) for galaxies of the specified `mass1` and `mass2` (in $M_\odot$). The `boxLength` should be set to an appropriate value to fully enclose (with sufficient buffering to allow for Fourier transformation) the two window functions. Must have the following interface:

```
 subroutine myImplementationWindowFunctions(self,mass1          , mass2,gridCount,&
&  boxLength,windowFunction1, windowFunction2)
  class           (surveyGeometryClass)                                    , &
&  intent(inout) :: self
  double precision                                                         , &
&  intent(in   ) :: mass1, mass2
  integer                                                                  , &
&  intent(in   ) :: gridCount
  double precision                                                         , &
&  intent(  out) :: boxLength
```

```
  complex         (c_double_complex   ), dimension(gridcount,gridcount,gridcount), &
& intent(  out) :: windowFunction1, windowFunction2
 end subroutine myImplementationWindowFunctions
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(surveyGeometryClass), intent(inout) :: self
 class(surveyGeometryClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (surveyGeometryClass), intent(inout) :: self
 integer                     , intent(in   ) :: stateFile
 type   (fgsl_file          ), intent(in   ) :: fgslStateFile
 integer(c_size_t           ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

`distanceMinimum` Returns the minimum distance (in Mpc) at which a galaxy of the specified `mass` (in $M_\odot$) would be included in the survey. A default implementation exists. If overridden the following interface must be used:

```
double precision function myImplementationDistanceMinimum(self,mass , magnitudeAbsolute&
&  , luminosity,field)
  class           (surveyGeometryClass), intent(inout)           :: self
  double precision                     , intent(in   ), optional :: mass, &
& magnitudeAbsolute, luminosity
  integer                              , intent(in   ), optional :: field
end double precision function myImplementationDistanceMinimum
```

`angularPowerAvailable` Returns true if angular power spectrum of survey window function is available. Must have the following interface:

```
logical function myImplementationAngularPowerAvailable(self)
 class(surveyGeometryClass), intent(inout) :: self
end logical function myImplementationAngularPowerAvailable
```

Existing implementations are:

`surveyGeometryRandomPoints` Implements survey geometries defined by random points.

`surveyGeometryLocalGroupClassical` Implements a geometry corresponding to the detectability of classical Local Group galaxies.

`surveyGeometryFullSky` Implements survey geometries over the full sky.

`surveyGeometryKelvin2014GAMAnear` Implements the geometry of the GAMAnear survey of Kelvin et al. [2014].

`surveyGeometryGunawardhana2013SDSS` Implements the geometry of the SDSS survey of Gunawardhana et al. [2013].

`surveyGeometryMonteroDorta2009SDSS` Implements the geometry of the SDSS survey of Montero-Dorta and Prada [2009].

`surveyGeometryBernardi2013SDSS` Implements the geometry of the SDSS survey of Bernardi et al. [2013]. See §12.59.2.

`surveyGeometryCaputi2011UKIDSSUDS` Implements the survey geometry of the SDSS sample used by Caputi et al. [2011]. See §12.59.9.

`surveyGeometryCombined` Implements a survey geometry which combines multiple other surveys.

`surveyGeometryMoustakas2013PRIMUS` Implements the geometry of the PRIMUS survey of Moustakas et al. [2013]. See §12.59.6.

`surveyGeometryBaldry2012GAMA` Implements the geometry of the GAMA survey of Baldry et al. [2012]. See §12.59.5.

`surveyGeometryLiWhite2009SDSS` Implements the survey geometry of the SDSS sample used by Li and White [2009]. See §12.59.1.

`surveyGeometryLocalGroupDES` Implements the geometry of the DES survey for Local Group dwarfs.

`surveyGeometryHearin2014SDSS` Implements the survey geometry of the SDSS sample used by Hearin et al. [2013].

`surveyGeometryDavidzon2013VIPERS` Implements the geometry of the VIPERS survey of Davidzon et al. [2013]. See §12.59.3.

`surveyGeometryMangle` Implements an abstract survey geometry using MANGLE polygons.

`surveyGeometryMartin2010ALFALFA` Implements the survey geometry of the SDSS sample used by Martin et al. [2010]. See §12.59.8.

`surveyGeometryTomczak2014ZFOURGE` Implements the geometry of the ZFOURGE survey of Tomczak et al. [2014]. See §12.59.4.

`surveyGeometryMuzzin2013ULTRAVISTA` Implements the geometry of the ULTRAVISTA survey of Muzzin et al. [2013].

`surveyGeometryLocalGroupSDSS` Implements the geometry of the SDSS survey with a depth for Local Group dwarf detection.

Tasks

Additional implementations for tasks are added using the `task` class. The implementation should be placed in a file containing the directive:

```
!# <task name="taskMyImplementation">
!# <description>A short description of the implementation.</description>
!# </task>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `taskMyImplementation`. The file *must* define a type that extends the `taskClass` class (or extends another type

which is itself an extension of the `taskClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
 class  (taskClass), intent(inout)           :: self
 logical           , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (taskClass      ), intent(inout)           :: self
 type   (inputParameters), intent(inout)           :: descriptor
 logical                 , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(taskClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`perform` Perform the task. Must have the following interface:

```
subroutine myImplementationPerform(self,status)
 class  (taskClass), intent(inout)           :: self
 integer           , intent(  out), optional :: status
end subroutine myImplementationPerform
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (taskClass), intent(inout) :: self
 integer           , intent(in   ) :: stateFile
 type   (fgsl_file), intent(in   ) :: fgslStateFile
 integer(c_size_t ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (taskClass     )                            , intent(inout) :: self
 type     (varying_string), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*         )                            , intent(in   ) :: sourceName
end subroutine myImplementationAllowedParameters
```

`requiresOutputFile` Should return true if the task requires the main output file to be open. A default implementation exists. If overridden the following interface must be used:

```
logical function myImplementationRequiresOutputFile(self)
 class(taskClass), intent(inout) :: self
end logical function myImplementationRequiresOutputFile
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(taskClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(taskClass), intent(inout) :: self
 class(taskClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (taskClass), intent(inout) :: self
 integer           , intent(in   ) :: stateFile
 type   (fgsl_file), intent(in   ) :: fgslStateFile
 integer(c_size_t ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`taskPosteriorSample` A task which performs sampling from a posterior distribution.

`taskBuildToolFSPS` A task which builds the FSPS tool.

`taskLocalGroupDatabase` A task which updates the Local Group database.

`taskExcursionSets` A task which computes and outputs the halo mass function and related quantities.

`taskBuildToolCloudy` A task which builds the Cloudy tool.

`taskEvolveForests` A task which evolves galaxies within a set of merger tree forests.

`taskConditionalMassFunction` A task which computes the conditional mass function in bins of mass for a fixed halo mass.

`taskMassFunctionCovariance` A task which computes and stores covariance matrices for mass functions.

`taskAGNSpectraHopkins2008BuildFile` A task which evolves galaxies within a set of merger tree forests.

`taskHaloModelProjectedCorrelationFunction` A task which generates a mock catalog of galaxies based on a simple halo model approach.

`taskCatalogProjectedCorrelationFunction` A task which computes projected correlation functions based on a simple halo model approach.

`taskMergerTreeFileBuilder` A task which computes and outputs the halo mass function and related quantities.

`taskNBodyAnalyze` A task which analyzes N-body simulation data.

`taskReport` A task which reports on version and build information.

`taskHaloMassFunction` A task which computes and outputs the halo mass function and related quantities.

`taskPowerSpectra` A task which computes and outputs the power spectrum and related quantities.

`taskIntergalacticMediumState` A task which outputs the state of the intergalactic medium.

`taskHaloModelGenerate` A task which generates a mock catalog of galaxies based on a simple halo model approach.

`taskBuildToolCAMB` A task which builds the CAMB tool.

`taskBuildToolRecFast` A task which builds the RecFast tool.

`taskHaloSpinDistribution` A task which computes and outputs the halo spin distribution.

`taskMulti` A task which performs multiple other tasks.

Tidal stripping in disks

Additional implementations for tidal stripping in disks are added using the `tidalStrippingDisks` class. The implementation should be placed in a file containing the directive:

```
!# <tidalStrippingDisks name="tidalStrippingDisksMyImplementation">
!# <description>A short description of the implementation.</description>
!# </tidalStrippingDisks>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `tidalStrippingDisksMyImplementation`. The file *must* define a type that extends the `tidalStrippingDisksClass` class (or extends another type which is itself an extension of the `tidalStrippingDisksClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (tidalStrippingDisksClass), intent(inout)          :: self
  logical                          , intent(in   ), optional :: includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (tidalStrippingDisksClass), intent(inout)           :: self
 type   (inputParameters         ), intent(inout)           :: descriptor
 logical                          , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`rateMassLoss` Returns the rate of mass loss (in $M_\odot$ Gyr^{-1}) due to tidal stripping of the disk component of `node`. Must have the following interface:

```
double precision function myImplementationRateMassLoss(self,node)
 class(tidalStrippingDisksClass), intent(inout) :: self
 type (treeNode                ), intent(inout) :: node
end double precision function myImplementationRateMassLoss
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(tidalStrippingDisksClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (tidalStrippingDisksClass), intent(inout) :: self
 integer                          , intent(in   ) :: stateFile
 type   (fgsl_file                ), intent(in   ) :: fgslStateFile
 integer(c_size_t                 ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (tidalStrippingDisksClass)                              , intent(inout) :: self
 type     (varying_string          ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                   )                              , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(tidalStrippingDisksClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(tidalStrippingDisksClass), intent(inout) :: self
 class(tidalStrippingDisksClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (tidalStrippingDisksClass), intent(inout) :: self
 integer                          , intent(in   ) :: stateFile
 type   (fgsl_file               ), intent(in   ) :: fgslStateFile
 integer(c_size_t                ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`tidalStrippingDisksNull` A null model of tidal stripping in galactic disks.

`tidalStrippingDisksSimple` A simple model of tidal stripping in galactic disks.

Tidal stripping in spheroids

Additional implementations for tidal stripping in spheroids are added using the `tidalStrippingSpheroids` class. The implementation should be placed in a file containing the directive:

```
!# <tidalStrippingSpheroids name="tidalStrippingSpheroidsMyImplementation">
!# <description>A short description of the implementation.</description>
!# </tidalStrippingSpheroids>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `tidalStrippingSpheroidsMyImplementation`. The file *must* define a type that extends the `tidalStrippingSpheroidsClass` class (or extends another type which is itself an extension of the `tidalStrippingSpheroidsClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
 class  (tidalStrippingSpheroidsClass), intent(inout)           :: self
 logical                              , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (tidalStrippingSpheroidsClass), intent(inout)           :: self
 type   (inputParameters             ), intent(inout)           :: descriptor
 logical                               , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`rateMassLoss` Returns the rate of mass loss (in $M_\odot$ Gyr^{-1}) due to tidal stripping of the spheroid component of `node`. Must have the following interface:

```
double precision function myImplementationRateMassLoss(self,node)
 class(tidalStrippingSpheroidsClass), intent(inout) :: self
 type (treeNode                    ), intent(inout) :: node
end double precision function myImplementationRateMassLoss
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(tidalStrippingSpheroidsClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (tidalStrippingSpheroidsClass), intent(inout) :: self
 integer                              , intent(in   ) :: stateFile
 type   (fgsl_file                    ), intent(in   ) :: fgslStateFile
 integer(c_size_t                     ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (tidalStrippingSpheroidsClass)                          , intent(inout) :: &
&  self
 type     (varying_string             ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                      )                          , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(tidalStrippingSpheroidsClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(tidalStrippingSpheroidsClass), intent(inout) :: self
 class(tidalStrippingSpheroidsClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (tidalStrippingSpheroidsClass), intent(inout) :: self
 integer                              , intent(in   ) :: stateFile
 type   (fgsl_file                    ), intent(in   ) :: fgslStateFile
 integer(c_size_t                     ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`tidalStrippingSpheroidsNull` A null model of tidal stripping in galactic spheroids.

`tidalStrippingSpheroidsSimple` A simple model of tidal stripping in galactic spheroids.

Transfer Function

Additional implementations for transfer function are added using the `transferFunction` class. The implementation should be placed in a file containing the directive:

```
!# <transferFunction name="transferFunctionMyImplementation">
!# <description>A short description of the implementation.</description>
!# </transferFunction>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `transferFunctionMyImplementation`. The file *must* define a type that extends the `transferFunctionClass` class (or extends another type which is itself an extension of the `transferFunctionClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
 class  (transferFunctionClass), intent(inout)           :: self
 logical                       , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (transferFunctionClass), intent(inout)           :: self
 type   (inputParameters      ), intent(inout)           :: descriptor
 logical                       , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(transferFunctionClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`value` Return the transfer function for $k =$`wavenumber` [Mpc^{-1}]. Must have the following interface:

```
double precision function myImplementationValue(self,wavenumber)
 class           (transferFunctionClass), intent(inout) :: self
 double precision                       , intent(in   ) :: wavenumber
end double precision function myImplementationValue
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (transferFunctionClass), intent(inout) :: self
 integer                       , intent(in   ) :: stateFile
 type   (fgsl_file            ), intent(in   ) :: fgslStateFile
 integer(c_size_t             ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`logarithmicDerivative` Return the logarithmic derivative of the transfer function for $k =$`wavenumber` [Mpc^{-1}]. Must have the following interface:

```
double precision function myImplementationLogarithmicDerivative(self,wavenumber)
 class           (transferFunctionClass), intent(inout) :: self
 double precision                       , intent(in   ) :: wavenumber
end double precision function myImplementationLogarithmicDerivative
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (transferFunctionClass)                          , intent(inout) :: self
 type     (varying_string       ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                 )                          , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`halfModeMass` Return the mass (in $M_\odot$) corresponding to the wavenumber at which the transfer function is suppressed by a factor of two due to small-scale dark matter particle physics. Must have the following interface:

```
double precision function myImplementationHalfModeMass(self,status)
 class  (transferFunctionClass), intent(inout)           :: self
 integer                       , intent(  out), optional :: status
end double precision function myImplementationHalfModeMass
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(transferFunctionClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(transferFunctionClass), intent(inout) :: self
 class(transferFunctionClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`epochTime` Return the cosmic time corresponding to the epoch for which this transfer function is defined. Must have the following interface:

```
double precision function myImplementationEpochTime(self)
 class(transferFunctionClass), intent(inout) :: self
end double precision function myImplementationEpochTime
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (transferFunctionClass), intent(inout) :: self
 integer                       , intent(in   ) :: stateFile
 type   (fgsl_file             ), intent(in   ) :: fgslStateFile
 integer(c_size_t              ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`transferFunctionAccelerator` A transfer function class which accelerates calculations of another transfer function class by tabulation for rapid interpolation.

`transferFunctionEisensteinHu1999` Provides the Eisenstein and Hu [1999] fitting function to the transfer function. The effective number of neutrino species and the summed mass (in electron volts) of all neutrino species are specified via the `neutrinoNumberEffective` and `neutrinoMassSummed` parameters respectively.

`transferFunctionCAMB` Provides a transfer function class in which the transfer function is generated by CAMB using the specified cosmological parameters. The transfer function is written out to a file in the `data/` directory and will be re-read later if needed.

`transferFunctionBode2001` Provides a transfer function based on the thermal WDM modifier of Bode et al. [2001].

`transferFunctionBBKSWDM` Provides a transfer function based on the WDM modifier of Bardeen et al. [1986].

`transferFunctionFile` Provides a transfer function from a tabulation given in an HDF5 file with the following structure:

```
HDF5 "transferFunction.hdf5" {
GROUP "/" {
ATTRIBUTE "description" {
DATATYPE  H5T_STRING {
STRSIZE 71;
STRPAD H5T_STR_NULLTERM;
CSET H5T_CSET_ASCII;
CTYPE H5T_C_S1;
}
DATASPACE  SCALAR
}
ATTRIBUTE "fileFormat" {
DATATYPE  H5T_STD_I32LE
DATASPACE  SCALAR
}
ATTRIBUTE "redshift" {
DATATYPE  H5T_STD_I32LE
DATASPACE  SCALAR
}
GROUP "extrapolation" {
GROUP "wavenumber" {
ATTRIBUTE "high" {
DATATYPE  H5T_STRING {
STRSIZE 11;
STRPAD H5T_STR_NULLTERM;
CSET H5T_CSET_ASCII;
CTYPE H5T_C_S1;
}
DATASPACE  SCALAR
}
ATTRIBUTE "low" {
DATATYPE  H5T_STRING {
STRSIZE 3;
STRPAD H5T_STR_NULLTERM;
CSET H5T_CSET_ASCII;
CTYPE H5T_C_S1;
}
DATASPACE  SCALAR
}
}
}
GROUP "parameters" {
ATTRIBUTE "HubbleConstant" {
DATATYPE  H5T_STRING {
```

```
STRSIZE 4;
STRPAD H5T_STR_NULLTERM;
CSET H5T_CSET_ASCII;
CTYPE H5T_C_S1;
}
DATASPACE  SCALAR
}
ATTRIBUTE "OmegaBaryon" {
DATATYPE  H5T_STRING {
STRSIZE 6;
STRPAD H5T_STR_NULLTERM;
CSET H5T_CSET_ASCII;
CTYPE H5T_C_S1;
}
DATASPACE  SCALAR
}
ATTRIBUTE "OmegaDarkEnergy" {
DATATYPE  H5T_STRING {
STRSIZE 5;
STRPAD H5T_STR_NULLTERM;
CSET H5T_CSET_ASCII;
CTYPE H5T_C_S1;
}
DATASPACE  SCALAR
}
ATTRIBUTE "OmegaMatter" {
DATATYPE  H5T_STRING {
STRSIZE 5;
STRPAD H5T_STR_NULLTERM;
CSET H5T_CSET_ASCII;
CTYPE H5T_C_S1;
}
DATASPACE  SCALAR
}
}
DATASET "transferFunction" {
DATATYPE  H5T_IEEE_F64LE
DATASPACE  SIMPLE { ( 1000 ) / ( 1000 ) }
}
DATASET "wavenumber" {
DATATYPE  H5T_IEEE_F64LE
DATASPACE  SIMPLE { ( 1000 ) / ( 1000 ) }
}
}
}
```

`transferFunctionIdentity` Provides an identity transfer function.

`transferFunctionBBKS` Provides the Bardeen et al. [1986] fitting function for the transfer function.

Unevolved Subhalo Mass Function

Additional implementations for unevolved subhalo mass function are added using the `unevolvedSubhaloMassFunction` class. The implementation should be placed in a file containing the directive:

```
!# <unevolvedSubhaloMassFunction name="unevolvedSubhaloMassFunctionMyImplementation">
!# <description>A short description of the implementation.</description>
!# </unevolvedSubhaloMassFunction>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `unevolvedSubhaloMassFunctionMyImplementation`. The file *must* define a type that extends the `unevolvedSubhaloMassFunctionClass` class (or extends another type which is itself an extension of the `unevolvedSubhaloMassFunctionClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
  class  (unevolvedSubhaloMassFunctionClass), intent(inout)          :: self
  logical                                   , intent(in   ), optional :: &
&  includeSourceDigest
 end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (unevolvedSubhaloMassFunctionClass), intent(inout)          :: self
 type   (inputParameters                  ), intent(inout)          :: descriptor
 logical                                   , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(unevolvedSubhaloMassFunctionClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (unevolvedSubhaloMassFunctionClass), intent(inout) :: self
 integer                                   , intent(in   ) :: stateFile
 type   (fgsl_file                        ), intent(in   ) :: fgslStateFile
 integer(c_size_t                         ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`integrated` Return the unevolved subhalo mass function per host at `time` [Gyr] in hosts of mass `massHost` [$M_\odot$] integrated between `massLow` and `massHigh` [$M_\odot$]. Must have the following interface:

```
 double precision function myImplementationIntegrated(self,time, massLow, massHigh, &
&  massHost)
  class           (unevolvedSubhaloMassFunctionClass), intent(inout) :: self
  double precision                                   , intent(in   ) :: time, massLow, &
&  massHigh, massHost
 end double precision function myImplementationIntegrated
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
  class    (unevolvedSubhaloMassFunctionClass)                          , intent(inout)&
&   :: self
  type     (varying_string                  ), allocatable, dimension(:), intent(inout)&
&   :: allowedParameters
  character(len=*                           )                          , intent(in   )&
&   :: sourceName
 end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationObjectType(self)
  class(unevolvedSubhaloMassFunctionClass), intent(inout) :: self
 end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationDeepCopy(self,destination)
  class(unevolvedSubhaloMassFunctionClass), intent(inout) :: self
  class(unevolvedSubhaloMassFunctionClass), intent(inout) :: destination
 end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
  class  (unevolvedSubhaloMassFunctionClass), intent(inout) :: self
  integer                                   , intent(in   ) :: stateFile
  type   (fgsl_file                        ), intent(in   ) :: fgslStateFile
  integer(c_size_t                         ), intent(in   ) :: stateOperationID
 end subroutine myImplementationStateRestore
```

`differential` Return the differential unevolved subhalo mass function per halo for `mass` [$M_\odot$] subhalos in `massHost` [$M_\odot$] hosts at `time` [Gyr]. Must have the following interface:

```
 double precision function myImplementationDifferential(self,time, mass, massHost)
  class           (unevolvedSubhaloMassFunctionClass), intent(inout) :: self
  double precision                                   , intent(in   ) :: time, mass, &
&  massHost
 end double precision function myImplementationDifferential
```

Existing implementations are:

`unevolvedSubhaloMassFunctionGiocoli2008` The halo mass function is computed from the function given by Giocoli et al. [2008].

Universe Operators

Additional implementations for universe operators are added using the `universeOperator` class. The implementation should be placed in a file containing the directive:

```
!# <universeOperator name="universeOperatorMyImplementation">
!# <description>A short description of the implementation.</description>
!# </universeOperator>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `universeOperatorMyImplementation`. The file *must* define a type that extends the `universeOperatorClass` class (or extends another type which is itself an extension of the `universeOperatorClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`operate` Operate on the universe. Must have the following interface:

```
subroutine myImplementationOperate(self,universe_)
 class(universeOperatorClass), intent(inout) :: self
 type (universe             ), intent(inout) :: universe_
end subroutine myImplementationOperate
```

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
 class  (universeOperatorClass), intent(inout)           :: self
 logical                       , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (universeOperatorClass), intent(inout)           :: self
 type   (inputParameters      ), intent(inout)           :: descriptor
 logical                       , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(universeOperatorClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (universeOperatorClass), intent(inout) :: self
 integer                       , intent(in   ) :: stateFile
 type   (fgsl_file            ), intent(in   ) :: fgslStateFile
 integer(c_size_t             ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (universeOperatorClass)                            , intent(inout) :: self
 type     (varying_string       ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                 )                            , intent(in   ) :: &
&  sourceName
 end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(universeOperatorClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(universeOperatorClass), intent(inout) :: self
 class(universeOperatorClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (universeOperatorClass), intent(inout) :: self
 integer                       , intent(in   ) :: stateFile
 type   (fgsl_file            ), intent(in   ) :: fgslStateFile
 integer(c_size_t             ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`universeOperatorIdentity` An identity operator on universes.

`universeOperatorIntergalacticMediumStateEvolve` An operator on universes which attaches hooks to compute evolution of the intergalactic medium.

Virial Density Contrasts

Additional implementations for virial density contrasts are added using the `virialDensityContrast` class. The implementation should be placed in a file containing the directive:

```
!# <virialDensityContrast name="virialDensityContrastMyImplementation">
!# <description>A short description of the implementation.</description>
!# </virialDensityContrast>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `virialDensityContrastMyImplementation`. The file *must* define a type that extends the `virialDensityContrastClass` class (or extends another type which is itself an extension of the `virialDensityContrastClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
 class  (virialDensityContrastClass), intent(inout)           :: self
 logical                            , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (virialDensityContrastClass), intent(inout)           :: self
 type   (inputParameters           ), intent(inout)           :: descriptor
 logical                            , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(virialDensityContrastClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`isMassDependent` Returns true if the virial density contrast is mass-dependent. A default implementation exists. If overridden the following interface must be used:

```
logical function myImplementationIsMassDependent(self)
 class(virialDensityContrastClass), intent(inout) :: self
end logical function myImplementationIsMassDependent
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (virialDensityContrastClass), intent(inout) :: self
 integer                            , intent(in   ) :: stateFile
 type   (fgsl_file                  ), intent(in   ) :: fgslStateFile
 integer(c_size_t                   ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (virialDensityContrastClass)                          , intent(inout) :: &
&  self
 type     (varying_string           ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*                    )                          , intent(in   ) :: &
&  sourceName
end subroutine myImplementationAllowedParameters
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationObjectType(self)
 class(virialDensityContrastClass), intent(inout) :: self
end type(varying_string) function myImplementationObjectType
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDeepCopy(self,destination)
 class(virialDensityContrastClass), intent(inout) :: self
 class(virialDensityContrastClass), intent(inout) :: destination
end subroutine myImplementationDeepCopy
```

`turnAroundOverVirialRadii` Returns the ratio of turnaround and virial radii at the given epoch. A default implementation exists. If overridden the following interface must be used:

```
double precision function myImplementationTurnAroundOverVirialRadii(self,time      , &
&  expansionFactor,collapsing)
 class           (virialDensityContrastClass), intent(inout)           :: self
 double precision                            , intent(in   ), optional :: time, &
&  expansionFactor
 logical                                     , intent(in   ), optional :: collapsing
end double precision function myImplementationTurnAroundOverVirialRadii
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (virialDensityContrastClass), intent(inout) :: self
 integer                            , intent(in   ) :: stateFile
 type   (fgsl_file                  ), intent(in   ) :: fgslStateFile
 integer(c_size_t                   ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

densityContrastRateOfChange Returns the rate of change of virial density contrast at the given epoch. Must have the following interface:

```
 double precision function myImplementationDensityContrastRateOfChange(self,mass,time   &
&     , expansionFactor,collapsing)
  class           (virialDensityContrastClass), intent(inout)           :: self
  double precision                            , intent(in   )           :: mass
  double precision                            , intent(in   ), optional :: time, &
&  expansionFactor
  logical                                     , intent(in   ), optional :: collapsing
 end double precision function myImplementationDensityContrastRateOfChange
```

densityContrast Returns the virial density contrast at the given epoch. Must have the following interface:

```
 double precision function myImplementationDensityContrast(self,mass,time      , &
&  expansionFactor,collapsing)
  class           (virialDensityContrastClass), intent(inout)           :: self
  double precision                            , intent(in   )           :: mass
  double precision                            , intent(in   ), optional :: time, &
&  expansionFactor
  logical                                     , intent(in   ), optional :: collapsing
 end double precision function myImplementationDensityContrast
```

Existing implementations are:

virialDensityContrastSphericalCollapseMatterLambda Dark matter halo virial density contrasts based on the spherical collapse in a matter plus cosmological constant universe.

virialDensityContrastKitayamaSuto1996 Kitayama and Suto [1996] dark matter halo virial density contrasts.

virialDensityContrastFriendsOfFriends Dark matter halo virial density contrasts based on the friends-of-friends algorithm linking length.

virialDensityContrastBryanNorman1998 Bryan and Norman [1998] dark matter halo virial density contrasts.

virialDensityContrastSphericalCollapseMatterDE Dark matter halo virial density contrasts based on the spherical collapse in a matter plus dark eneryg universe.

virialDensityContrastPercolation Dark matter halo virial density contrasts based on the percolation analysis of More et al. [2011].

virialDensityContrastFixed Fixed dark matter halo virial density contrasts.

Virial Orbits

Additional implementations for virial orbits are added using the virialOrbit class. The implementation should be placed in a file containing the directive:

```
!# <virialOrbit name="virialOrbitMyImplementation">
!# <description>A short description of the implementation.</description>
!# </virialOrbit>
```

where `MyImplementation` is an appropriate name for the implemention. This file should be treated as a regular Fortran module, but without the initial `module` and final `end module` lines. That is, it may contain `use` statements and variable declarations prior to the `contains` line, and should contain all functions required by the implementation after that line. Function names should begin with `virialOrbitMyImplementation`. The file *must* define a type that extends the `virialOrbitClass` class (or extends another type which is itself an extension of the `virialOrbitClass` class), containing any data needed by the implementation along with type-bound functions required by the implementation. The following type-bound functions are required (unless inherited from the parent type):

`hashedDescriptor` Return a hash of the descriptor for this object, optionally include the source code digest in the hash. A default implementation exists. If overridden the following interface must be used:

```
type(varying_string) function myImplementationHashedDescriptor(self,includeSourceDigest&
&  )
 class  (virialOrbitClass), intent(inout)          :: self
 logical                  , intent(in   ), optional :: includeSourceDigest
end type(varying_string) function myImplementationHashedDescriptor
```

`descriptor` Return an input parameter list descriptor which could be used to recreate this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationDescriptor(self,descriptor,includeMethod)
 class  (virialOrbitClass), intent(inout)          :: self
 type   (inputParameters ), intent(inout)          :: descriptor
 logical                  , intent(in   ), optional :: includeMethod
end subroutine myImplementationDescriptor
```

`autoHook` Insert any event hooks required by this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAutoHook(self)
 class(virialOrbitClass), intent(inout) :: self
end subroutine myImplementationAutoHook
```

`stateStore` Store the state of this object to file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateStore(self,stateFile,fgslStateFile,stateOperationID)
 class  (virialOrbitClass), intent(inout) :: self
 integer                  , intent(in   ) :: stateFile
 type   (fgsl_file       ), intent(in   ) :: fgslStateFile
 integer(c_size_t        ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateStore
```

`allowedParameters` Return a list of parameter names allowed for this object. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationAllowedParameters(self,allowedParameters,sourceName)
 class    (virialOrbitClass)                          , intent(inout) :: self
 type     (varying_string  ), allocatable, dimension(:), intent(inout) :: &
&  allowedParameters
 character(len=*           )                          , intent(in   ) :: sourceName
end subroutine myImplementationAllowedParameters
```

`velocityTangentialMagnitudeMean` Returns the mean of the magnitude of tangential velocity averaged over all orbits. Must have the following interface:

```
 double precision function myImplementationVelocityTangentialMagnitudeMean(self,node, &
&  host)
  class(virialOrbitClass), intent(inout) :: self
  type (treeNode        ), intent(inout) :: node, host
 end double precision function myImplementationVelocityTangentialMagnitudeMean
```

`objectType` Return the type of the object. A default implementation exists. If overridden the following interface must be used:

```
 type(varying_string) function myImplementationObjectType(self)
  class(virialOrbitClass), intent(inout) :: self
 end type(varying_string) function myImplementationObjectType
```

`velocityTangentialVectorMean` Returns the mean vector of the vector tangential velocity averaged over all orbits. Must have the following interface:

```
 double precision, dimension(3) function myImplementationVelocityTangentialVectorMean(&
&  self,node, host)
  class(virialOrbitClass), intent(inout) :: self
  type (treeNode        ), intent(inout) :: node, host
 end double precision, dimension(3) function &
&  myImplementationVelocityTangentialVectorMean
```

`orbit` Returns an orbit object. Must have the following interface:

```
 type(keplerOrbit) function myImplementationOrbit(self,node               , host,&
&  acceptUnboundOrbits)
  class  (virialOrbitClass), intent(inout) :: self
  type   (treeNode        ), intent(inout) :: node, host
  logical                  , intent(in   ) :: acceptUnboundOrbits
 end type(keplerOrbit) function myImplementationOrbit
```

`deepCopy` Perform a deep copy of the object. A default implementation exists. If overridden the following interface must be used:

```
 subroutine myImplementationDeepCopy(self,destination)
  class(virialOrbitClass), intent(inout) :: self
  class(virialOrbitClass), intent(inout) :: destination
 end subroutine myImplementationDeepCopy
```

`velocityTotalRootMeanSquared` Returns the square root of the mean of the squared total velocity averaged over all orbits. Must have the following interface:

```
 double precision function myImplementationVelocityTotalRootMeanSquared(self,node, host)
  class(virialOrbitClass), intent(inout) :: self
  type (treeNode        ), intent(inout) :: node, host
 end double precision function myImplementationVelocityTotalRootMeanSquared
```

`densityContrastDefinition` Returns a `virialDensityContrast` object describing the virial density contrast used to define this orbit class. Must have the following interface:

```
 class(virialDensityContrastClass) function myImplementationDensityContrastDefinition(&
&  self)
  class(virialOrbitClass), intent(inout) :: self
 end class(virialDensityContrastClass) function &
&  myImplementationDensityContrastDefinition
```

`stateRestore` Restore the state of this object from file. A default implementation exists. If overridden the following interface must be used:

```
subroutine myImplementationStateRestore(self,stateFile,fgslStateFile,stateOperationID)
 class  (virialOrbitClass), intent(inout) :: self
 integer                  , intent(in   ) :: stateFile
 type   (fgsl_file       ), intent(in   ) :: fgslStateFile
 integer(c_size_t        ), intent(in   ) :: stateOperationID
end subroutine myImplementationStateRestore
```

Existing implementations are:

`virialOrbitBenson2005` Virial orbits using the Benson [2005] orbital parameter distribution. See §12.51.2.

`virialOrbitWetzel2010` Virial orbits using the Wetzel [2010] orbital parameter distribution. See §12.51.2.

`virialOrbitJiang2014` Virial orbits using the Jiang et al. [2014] orbital parameter distribution. See §12.51.2.

`virialOrbitFixed` Virial orbits assuming fixed orbital parameters. See §12.51.2.

`virialOrbitSpinCorrelated` A virial orbit class which assigns infall directions and tangential velocities directions to an orbit return by another virial orbit class such that the orbital angular momentum of the satellite is correlated with the spin vector of the host. Specifically, the angle, θ, between the orbital angular momentum of the satellite and the spin of the host is assumed to be distributed such that $P(\cos\theta) = (1 + \alpha|\lambda|\cos\theta)/2$, where $|\lambda|$ is the magnitude of the host halo spin, and α is a parameter.

`virialOrbitIsotropic` Virial orbits which assumes an isotropic distribution of infall directions and tangential velocities applied to another virial orbit class.

Accretion Disks

Additional methods for accretion disk properties can be added using the `accretionDisksMethod` directive. The directive should contain a single argument, giving the name of a subroutine to be called to initialize the method. For example, the `Shakura-Sunyae` method is described by a directive:

```
!# <accretionDisksMethod>
!#  <unitName>Accretion_Disks_Shakura_Sunyaev_Initialize</unitName>
!# </accretionDisksMethod>
```

Here, `Accretion_Disks_Shakura_Sunyaev_Initialize` is the name of a subroutine which will be called to initialize the method. The initialization subroutine must have the following form:

```
 subroutine Method_Initialize(accretionDisksMethod,Accretion_Disk_Radiative_Efficiency_Get,Black_Hole_S
   implicit none
   type(varying_string),          intent(in)    :: accretionDisksMethod
   procedure(),          pointer, intent(inout) :: Accretion_Disk_Radiative_Efficiency_Get,Black_Hole_S
```

```
    if (accretionDisksMethod == 'myMethod') then
       Accretion_Disk_Radiative_Efficiency_Get => My_Accretion_Disk_Radiative_Efficiency_Get
       Black_Hole_Spin_Up_Rate_Get             => My_Black_Hole_Spin_Up_Rate_Get
       Accretion_Disk_Jet_Power_Get            => My_Accretion_Disk_Jet_Power_Get
    end if
    return
  end subroutine Method_Initialize
```

where `myMethod` is the name of this method as will be specified by the `accretionDisksMethod` input parameter. The procedure pointers `Accretion_Disk_Radiative_Efficiency_Get`, `Black_Hole_Spin_Up_Rate_Get` and `Accretion_Disk_Jet_Power_Get` must be set to point to functions which return the radiative efficiency, black hole spin up rate and jet power for the accretion disk respectively as described below. The initialization subroutine should perform any other tasks required to initialize the module (such as reading parameters etc.).

The radiative efficiency function must have the form:

```
 double precision function My_Accretion_Disk_Radiative_Efficiency_Get(thisNode,massAccretionRate)
    implicit none
    type(treeNode),   intent(inout), pointer :: thisNode
    double precision, intent(in)             :: massAccretionRate
    .
    .
    .
    return
 end function My_Accretion_Disk_Radiative_Efficiency_Get
```

The function must return the radiative efficiency for the accretion disk in `thisNode`. The black hole spin function must have the form:

```
 double precision function My_Black_Hole_Spin_Up_Rate_Get(thisNode,massAccretionRate)
    implicit none
    type(treeNode),   intent(inout), pointer :: thisNode
    double precision, intent(in)             :: massAccretionRate
    .
    .
    .
    return
 end function My_Black_Hole_Spin_Up_Rate_Get
```

The function must return the spin-up rate for the black hole in `thisNode` given the `massAccretionRate`. The jet power function must have the form:

```
 double precision function My_Accretion_Disk_Jet_Power_Get(thisNode,massAccretionRate)
    implicit none
    type(treeNode),   intent(inout), pointer :: thisNode
    double precision, intent(in)             :: massAccretionRate
    .
    .
    .
    return
 end function My_Accretion_Disk_Jet_Power_Get
```

The function must return (in units of $M_\odot$ (km/s)^2 Gyr^{-1}) the jet power for the black hole/accretion disk system in `thisNode` given the `massAccretionRate`.

Currently defined accretion disk methods are:

`Shakura-Sunyaev` Computes the properties of a thin, radiatively efficiency accretion disk.

`ADAF` Computes the properties of an ADAF using the model of Benson and Babul [2009].

`switched` Select either `Shakura-Sunyaev` or `ADAF` accretion disks based on the accretion rate:

$$\begin{aligned} \dot{m}_{\rm minimum} < \dot{M}_{\bullet,0}/\dot{M}_{\rm Eddington} < \dot{m}_{\rm maximum} &\rightarrow \text{Shakura-Sunyaev} \\ \text{otherwise} &\rightarrow \text{ADAF}, \end{aligned} \tag{16.10}$$

where $\dot{m}_{\rm minimum}$=`accretionRateThinDiskMinimum` and $\dot{m}_{\rm maximum}$=`accretionRateThinDiskMaximum` are input parameters.

`eddingtonLimited` Assumes no specific disk structure, instead setting the radiative efficiency to a fixed number and the jet power to a fixed fraction of the Eddington luminosity.

Analysis

Additional methods for on-the-fly analysis of merger trees can be added using the `mergerTreeAnalysisTask` directive. The directive should contain a single argument, giving the name of a subroutine to be called to initialize the method. For example, the mass distributions method is described by a directive:

```
!# <mergerTreeAnalysisTask>
!#  <unitName>Galacticus_Output_Analysis_Mass_Dpndnt_Sz_Dstrbtins</unitName>
!# </mergerTreeAnalysisTask>
```

Here, `Galacticus_Output_Analysis_Mass_Dpndnt_Sz_Dstrbtins` is the name of a subroutine which will be called to perform the analysis. The analysis subroutine must have the following form:

```
subroutine Tree_Analyze(thisTree,thisNode,iOutput,mergerTreeAnalyses)
  implicit none
  type            (mergerTree    ), intent(in   )                 :: thisTree
  type            (treeNode      ), intent(inout), pointer        :: thisNode
  integer                          , intent(in   )                 :: iOutput
  type            (varying_string), intent(in   ), dimension(:  ) :: mergerTreeAnalyses
  .
  .
  .
  return
end subroutine Method_Initialize
```

This function will be called once for each node in each tree being output. The function is passed the merger tree object as `thisTree`, along with a pointer to the node to be analyzed as `thisNode`. Additionally, the current output number is passed as `iOutput`. Finally, a list of analyses that were requested (by user input) to be performed is given in the `mergerTreeAnalyses` array. The function should check if one or more of the entries in `mergerTreeanalyses` correspond to analyses that it performs. (Note that this need only be done on the first call to this function—the values of `mergerTreeAnalyses` will not change between calls.) If an analysis is matched in this way it should be performed. Typically, an analysis function might accumulate the results of analysis and then finalize and output them prior to completion of the Galacticus model through the use of a `hdfPreCloseTask` (see §16.4.3).

Currently defined merger tree analysis methods are (see §4.4):

mass functions Constructs mass functions for a variety of different surveys. Currently supported analysis names are: `sdssStellarMassFunctionZ0.07`, `alfalfaHiMassFunctionZ0.00`, `primusStellarMassFunctionZ0.100`, `primusStellarMassFunctionZ0.250`, `primusStellarMassFunctionZ0.350`, `primusStellarMassFunctionZ0.450`, `primusStellarMassFunctionZ0.575`, `primusStellarMassFunctionZ0.725`, and `primusStellarMassFunctionZ0.900` Each such analysis is defined internally to this module through the appropriate redshift, binning, random and systematic errors, and arbitrary mapping of component masses into observed masses. The covariance matrix of the mass function is also computed (see §9.5.3 for details).

mass-dependent size distributions Constructs a two-dimensional histogram of galaxy sizes in bins of stellar mass. Currently supported analyses are `sdssSizeFunctionZ0.07`. Each such analysis is defined internally to this module through the appropriate redshift, binning, random and systematic errors, and arbitrary mapping of component masses/radii into observed masses/radii. The covariance matrix of the size function is also computed (see §9.5.3 for details).

Radiation Components

Radiation components (i.e. types of radiation field that may be added to any radiation object; see §16.5.3) are defined using a combination of several directives: `radiationLabel`, `radiationSet`, `radiationTemperature` and `radiationFlux`. For example, the cosmic microwave background radiation component is defined by the following set of directives:

```
!# <radiationLabel>
!#  <label>CMB</label>
!# </radiationLabel>

!# <radiationSet>
!#  <unitName>Radiation_Set_CMB</unitName>
!#  <label>CMB</label>
!# </radiationSet>

!# <radiationTemperature>
!#  <unitName>Radiation_Temperature_CMB</unitName>
!#  <label>CMB</label>
!# </radiationTemperature>

!# <radiationFlux>
!#  <unitName>Radiation_Flux_CMB</unitName>
!#  <label>CMB</label>
!# </radiationFlux>
```

The first of these, `radiationLabel`, should contain a single element, `label`, which gives a label that will be used to identify this component, both in other directives and also in the internal parameters used to select this radiation component (e.g. in this case, a parameter `radiationTypeCMB` will be available within Galacticus to select the cosmic microwave background component). The other directives must all specify the same `label` element and additional give, in a `unitName` element, the name of a function/subroutine to be called to perform the relevant calculation.

The `radiationSet` directive must specify a subroutine with the following template:

```
subroutine Radiation_Set(componentMatched,thisNode,radiationProperties)
 implicit none
 logical,          intent(in)                                  :: componentMatched
```

```
 type(treeNode),   intent(inout), pointer                   :: thisNode
 double precision, intent(inout), allocatable, dimension(:) :: radiationProperties

 if (.not.componentMatched) return
 .
 .
 .
 return
end subroutine Radiation_Set
```

If `componentMatched` is true, then the subroutine should set the radiation component, otherwise it should exit immediately. If the radiation component is to be set, then the routine can allocate the `radiationProperties` array as necessary to store any data needed to specify the radiation field. These data should then be set using, if necessary, any relevant information from `thisNode`.

The `radiationTemperature` directive should specify a subroutine with the following template:

```
subroutine Radiation_Temperature(requestedType,ourType,radiationProperties,radiationTemperature,radiat
 implicit none
 integer,          intent(in)                           :: requestedType,ourType
 double precision, intent(in),              dimension(:) :: radiationProperties
 double precision, intent(inout)                        :: radiationTemperature
 integer,          intent(in),    optional, dimension(:) :: radiationType

 if (requestedType /= ourType) return
 if (present(radiationType)) then
    if (all(radiationType /= ourType)) return
 end if
 .
 .
 .
 return
end subroutine Radiation_Temperature
```

The tests in the above should always be included so that the subroutine exits immediately if the component type is not active or not requested. Once these tests have been made, the subroutine should set the temperature (in units of Kelvin) of the radiation field (if applicable).

The `radiationFlux` directive should specify a subroutine with the following template:

```
subroutine Radiation_Flux(requestedType,ourType,radiationProperties,wavelength,radiationFlux,radiatio
  implicit none
  integer,          intent(in)                           :: requestedType,ourType
  double precision, intent(in)                           :: wavelength
  double precision, intent(in),              dimension(:) :: radiationProperties
  double precision, intent(inout)                        :: radiationFlux
  integer,          intent(in),   optional, dimension(:) :: radiationType

  if (requestedType /= ourType) return
  if (present(radiationType)) then
     if (all(radiationType /= ourType)) return
  end if
  .
```

```
    .
    .
    return
  end subroutine Radiation_Flux
```

The tests in the above should always be included so that the subroutine exits immediately if the component type is not active or not requested. Once these tests have been made, the subroutine should add the flux (in units of ergs cm^2 s^{-1} Hz^{-1} ster^{-1}) at the specified `wavelength` (in units of Å) of the radiation field to that in `radiationFlux`.

Currently defined radiation component types are:

`null` A null component with no radiation.

`CMB` The cosmic microwave background, assumed to be a perfect blackbody spectrum with a temperature equal to `[T_CMB]`$(1+z)$.

`IGB` The intergalactic background light, set using the method selected by `[radiationIntergalacticBackgroundMethod]`; see §16.4.1.

Radiation Components: Intergalactic Background

Additional methods for the intergalactic background radiation component can be added using the `radiationIntergalacticBa` directive. The directive should contain a single argument, giving the name of a subroutine to be called to initialize the method. For example, the `file` method is described by a directive:

```
!# <radiationIntergalacticBackgroundMethod>
!#  <unitName>Radiation_IGB_File_Initialize</unitName>
!# </radiationIntergalacticBackgroundMethod>
```

Here, `Radiation_IGB_File_Initialize` is the name of a subroutine which will be called to initialize the method. The initialization subroutine must have the following form:

```
  subroutine Method_Initialize(radiationIntergalacticBackgroundMethod,Radiation_Set_Intergalactic_Backgr
    implicit none
    type(varying_string),          intent(in)    :: radiationIntergalacticBackgroundMethod
    procedure(),          pointer, intent(inout) :: Radiation_Set_Intergalactic_Background_Do,Radiation_

    if (radiationIntergalacticBackgroundMethod == 'myMethod') then
      Radiation_Set_Intergalactic_Background_Do  => My_Method_Set
      Radiation_Flux_Intergalactic_Background_Do => My_Method_Flux
    end if
    return
  end subroutine Method_Initialize
```

where `myMethod` is the name of this method as will be specified by the `radiationIntergalacticBackgroundMethod` input parameter. The procedure pointers `Radiation_Set_Intergalactic_Background_Do` and `Radiation_-Flux_Intergalactic_Background_Do` must be set to point to subroutines which set the radiation field and return its flux as described below. The initialization subroutine should perform any other tasks required to initialize the module (such as reading parameters etc.).

The set subroutine must have the form:

```
  subroutine My_Method_Set(thisNode,radiationProperties)
    implicit none
```

```
    type(treeNode),   intent(inout), pointer                   :: thisNode
    double precision, intent(inout), allocatable, dimension(:) :: radiationProperties

    return
  end subroutine My_Method_Set
```

and should set the radiation component as described in §16.4.1. The flux subroutine must have the form:

```
   subroutine My_Method_Flux(radiationProperties,wavelength,radiationFlux)
    implicit none
    double precision, intent(in)                 :: wavelength
    double precision, intent(in),   dimension(:) :: radiationProperties
    double precision, intent(inout)              :: radiationFlux

    return
   end subroutine My_Method_Flux
```

and should increment `radiationFlux` as described in §16.4.1.

Currently defined intergalactic background radiation methods are:

`file` The intergalatic background radiation field, specified as a function of cosmic time, is read from a file. The flux is determined by linearly interpolating to the required time and wavelength. The XML file to read is specified by `[radiationIGBFileName]`. An example of the required file structure is:

```
<spectrum>
  <URL>http://adsabs.harvard.edu/abs/1996ApJ...461...20H</URL>
  <description>Cosmic background radiation spectrum from quasars alone.</description>
  <reference>Haardt, F. &amp;amp; Madau, P. 1996, ApJ, 461, 20</reference>
  <source>Francesco Haardt on Aug 6 2005, via Cloudy 08.00</source>
  <wavelengths>
    <datum>0.0002481</datum>
    <datum>0.001489</datum>
    .
    .
    .
    <units>Angstroms</units>
  </wavelengths>
  <spectra>
    <datum>7.039E-49</datum>
    <datum>8.379E-48</datum>
    <datum>1.875E-39</datum>
    <datum>7.583E-38</datum>
    .
    .
    .
    <redshift>0</redshift>
    <units>erg cm^-2 s^-1 Hz^-1 sr^-1</units>
  </spectra>
</spectrum>
```

The optional `URL`, `description`, `reference` and `source` elements can be used to give the provenance of the data. The `wavelengths` element should contain a set of `datum` elements each containing a wavelength (in increasing order) at which the spectrum will be tabulated. Wavelengths must be given in Angstroms. Multiple `spectra` elements can be given, each specifying the spectrum at a redshift as given in the `redshift` element. Each `spectra` element must contain an array of `datum` elements that gives the spectrum at each wavelength listed in the `wavelength` element. Spectra must be in units of erg cm^{-2} s^{-1} Hz^{-1} sr^{-1}.

Tree Timing

Additional methods for tree timing (i.e. the time taken to process a given merger tree) can be added using the `timePerTreeMethod` directive. The directive should contain a single argument, giving the name of a subroutine to be called to initialize the method. For example, the `file` method is described by a directive:

```
!# <timePerTreeMethod>
!#  <unitName>Galacticus_Time_Per_Tree_File_Initialize</unitName>
!# </timePerTreeMethod>
```

Here, `Galacticus_Time_Per_Tree_File_Initialize` is the name of a subroutine which will be called to initialize the method. The initialization subroutine must have the following form:

```
subroutine Method_Initialize(timePerTreeMethod,Galacticus_Time_Per_Tree_Get)
  implicit none
  type(varying_string),          intent(in)    :: timePerTreeMethod
  procedure(),          pointer, intent(inout) :: Galacticus_Time_Per_Tree_Get

  if (timePerTreeMethod == 'myMethod') then
     Galacticus_Time_Per_Tree_Get => My_Time_Per_Tree_Get
     .
     .
     .
  end if
  return
end subroutine Method_Initialize
```

where `myMethod` is the name of this method as will be specified by the `timePerTreeMethod` input parameter. The procedure pointer `Galacticus_Time_Per_Tree_Get` must be set to point to a function which returns an estimate of the time taken (in seconds) to process a merger tree. The initialization subroutine should perform any other tasks required to initialize the module (such as reading parameters etc.).

The function must have the form:

```
 double precision function Time_Per_Tree(treeRootMass)
  implicit none
  double precision, intent(in) :: treeRootMass
  .
  .
  .
  return
 end function Time_Per_Tree
```

The function must return an estimate of the time taken (in seconds) to process a merger tree with the given `treeRootMass`.

Currently defined tree timing methods are:

`file` This method reads coefficients of a simple fitting formula for the processing time from a file, specified via the `[timePerTreeFitFileName]` parameter (see §17.2).

16.4.2. Events

Events are triggered during merger tree evolution. Examples are when a node needs to be promoted to its parent node, or when a minor node merges with its parent.

Node Promotion Events

Additional methods for node promotion (i.e. when a primary progenitor reaches its parent halo) can be added using the `nodePromotionTask` directive. The directive should contain a single argument, giving the name of a subroutine to be called to initialize the method. For example, the `basic` tree node method uses this directive as follows:

```
!# <nodePromotionTask>
!#  <unitName>Tree_Node_Basic_Promote</unitName>
!# </nodePromotionTask>
```

Here, `Tree_Node_Basic_Promote` is the name of a subroutine which will be called to perform whatever tasks are required prior to the promotion. The subroutine must have the following form:

```
 subroutine Node_Promotion_Task(thisNode)
  implicit none
  type(treeNode), pointer, intent(inout) :: thisNode
  .
  .
  .
  return
end subroutine Node_Promotion_Task
```

where `thisNode` is the node about to be promoted.

16.4.3. Tasks

Tasks are any processing which must be performed on a node as a result of some specific event (such as a merger).

Calculation Reset Tasks

Additional methods for calculation reset tasks (i.e. flagging that the properties of a node may have changed so that any calculations must be performed anew) can be added using the `calculationResetTask` directive. The directive should contain a single argument, giving the name of a subroutine to be called to perform the task. For example, the standard hot halo component adds a task as follows:

```
!# <calculationResetTask>
!#  <unitName>Tree_Node_Hot_Halo_Reset_Standard</unitName>
!# </calculationResetTask>
```

Here, `Tree_Node_Hot_Halo_Reset_Standard` is the name of a subroutine which will be called to perform whatever tasks are required. The subroutine must have the following form:

```
 subroutine Reset_Task(thisNode)
  implicit none
  type(treeNode), pointer, intent(inout) :: thisNode
  .
  .
  .
  return
end subroutine Prederivative_Task
```

where `thisNode` is the node for which derivatives will be computed. Tasks typically involve precomputing quantities that will be used in finding the derivatives or resetting the state so that stored quantities will be recomputed as needed.

Decode Property Identifier Tasks

Additional property identifier decoding tasks (i.e. determining the name of a property from a set of integer identifiers) can be added using the `decodePropertyIdentifiersTask` directive. The directive should contain a single argument, giving the name of a subroutine to be called to perform the task. For example, the Hernquist spheroid component adds a task as follows:

```
!# <decodePropertyIdentifiersTask>
!#  <unitName>Hernquist_Spheroid_Property_Identifiers_Decode</unitName>
!# </decodePropertyIdentifiersTask>
```

Here, `Hernquist_Spheroid_Property_Identifiers_Decode` is the name of a subroutine which will be called to perform the decoding task. The subroutine must have the following form:

```
 subroutine Property_Identifier_Decode_Task(propertyComponent,propertyObject,propertyIndex,matchedProp
  implicit none
  integer,              intent(in)    :: propertyComponent,propertyObject,propertyIndex
  logical,              intent(inout) :: matchedProperty
  type(varying_string), intent(inout) :: propertyName
  .
  .
  .
  return
end subroutine Property_Identifier_Decode_Task
```

The task should check whether `propertyComponent` matches its stored `componentIndex` value. If it does, it should set `propertyName` to a suitable name (e.g. `hernquistSpheroid::stellarMass`) and set `matchedProperty=true`. The value of `propertyObject` will be either `objectTypeProperty` indicating that the object in question is a standard property, or `objectTypeHistory` indicating that it is a history. The value of `propertyIndex` then gives the position of the object in question in the array of properties or histories.

Evolution Timestep Tasks

Merger tree nodes are evolved over some fixed timestep before evolution is stopped and other processing is allowed. The timestep is always sufficiently small such that the node does not evolve past the time of its parent node, nor does it evolve past the time of any of its satellite nodes. An arbitrary number of other criteria can be used to adjust the timestep. Such a criterion can be added using the `timeStepsTask` directive. For example, the `simple` timestep task adds itself using

```
!# <timeStepsTask>
!#  <unitName>Merger_Tree_Timestep_Simple</unitName>
!# </timeStepsTask>
```

Here, `unitName` gives the name of the subroutine to be called to (possibly) adjust the timestep. It should have the following form:

```
subroutine My_Timestep(thisNode,timeStep,End_Of_Timestep_Task,report,lockNode,lockType)
  implicit none
  type     (treeNode      ), intent(inout), pointer           :: thisNode
  procedure(              ), intent(inout), pointer           :: End_Of_Timestep_Task
  double precision         , intent(inout)                    :: timeStep
  logical                  , intent(in   )                    :: report
  type     (treeNode      ), intent(inout), pointer, optional :: lockNode
  type     (varying_string), intent(inout),          optional :: lockType
  .
  .
  .
  return
end subroutine My_Timestep
```

This subroutine should compute a suitable timestep for `thisNode` and, if it is less than the currently defined value of `timeStep` should set `timeStep` to that value. Optionally, the procedure pointer `End_Of_Timestep_Task` can be set to point to a subroutine which will be called after the node is evolved to the end of the timestep. It is acceptable for this pointer to be null. Note that the `End_Of_Timestep_Task` will only be called for the task which provided the shortest timestep—other tasks can always request to be called again when the next timestep is determined. The subroutine to be called at the end of the timestep must have the form:

```
subroutine My_End_Of_Timestep_Task(thisTree,thisNode,deadlockStatus)
  implicit none
  type(mergerTree), intent(in)             :: thisTree
  type(treeNode),   intent(inout), pointer :: thisNode
  integer,          intent(inout)          :: deadlockStatus
  .
  .
  .
  return
end subroutine My_End_Of_Timestep_Task
```

The `deadlockStatus` argument should be set to `isNotDeadlocked` (provided by the Merger_Trees_Evolve_Deadlock_Status module) if, and only if, the end of timestep task makes some change to the state of the tree (e.g. merging a node), to indicate that the tree was not deadlocked in this pass (i.e. something actually changed in the tree).

If the `report` argument is `true` then the function should report the value of `timestep` prior to exiting. (This is used in reporting on timestepping criteri in deadlocked trees.) It is recommended that the report be made using the Evolve_To_Time_Report() function. Additionally, if the optional `lockNode` and `lockType` arguments are present then additional information can be supplied to aid in diagnosing deadlock conditions. If the current task is limiting the timestep then the `lockNode` pointer should be set to point to whichever node is causing the limit (which may be `thisNode` or some other node, e.g. a satellite of `thisNode`, etc.), and `lockType` should be set to a short description label identifying the type of limit.

Galactic Component Density

The function `Galactic_Structure_Density()` computes the density of material at a given position within a node. To have their density counted, each component must register a task using:

```
!# <densityTask>
!#  <unitName>Density_Procedure</unitName>
!# </densityTask>
```

where `Density_Procedure` is the name of a function with the following template

```
double precision function Density_Procedure(thisNode,position,coordinateSystem,componentType,massType,w
   type(treeNode),   intent(inout), pointer  :: thisNode
   integer,          intent(in)              :: massType,coordinateSystem,componentType,weightBy,weight
   double precision, intent(in)              :: radius
   logical         , intent(in),    optional :: haloLoaded
   .
   .
   .
   return
end function Density_Procedure
```

If `componentType` is a match to the component then the function should return the density of the component matching type `massType` at `position` for `thisNode`. `componentType` and `massType` can take one of the values described in §16.2. In the above "density" can actually refer to different quantities depending on the values of `weightBy` (and `weightIndex`):

`weightByMass` The actual mass should be returned (the value of `weightIndex` is irrelevant);

`weightByLuminosity` The `weightIndex`$^{\text{th}}$ luminosity should be returned.

If `haloLoaded=true` (which should be the default if this option is not present), then the effects of baryonic loading on the halo profile should be taken into account where necessary. Otherwise, the effects of baryonic loading should be ignored.

Galactic Component Enclosed Mass

The function `Galactic_Structure_Enclosed_Mass()` computes the mass within a specified radius in a node. To have their mass counted, each component must register a task using:

```
!# <enclosedMassTask>
!#  <unitName>Enclosed_Mass_Procedure</unitName>
!# </enclosedMassTask>
```

where `Enclosed_Mass_Procedure` is the name of a function with the following template

```
double precision function Enclosed_Mass_Procedure(thisNode,radius,componentType,massType,weightBy,weigh
   type(treeNode),   intent(inout), pointer  :: thisNode
   integer,          intent(in)              :: massType,componentType,weightBy,weightIndex
   double precision, intent(in)              :: radius
   logical         , intent(in),    optional :: haloLoaded
   .
   .
   .
   return
end function Enclosed_Mass_Procedure
```

If `componentType` is a match to the component then the function should return the "mass" of the component matching type `massType` within `radius` for `thisNode`. `componentType` and `massType` can take one of the values described in §16.2. If `radius` is equal to or greater than `radiusLarge` the routine should the total "mass" (i.e. "mass" within infinite radius). In the above "mass" can actually refer to different quantities depending on the values of `weightBy` (and `weightIndex`):

`weightByMass` The actual mass should be returned (the value of `weightIndex` is irrelevant);

`weightByLuminosity` The `weightIndex`$^{\text{th}}$ luminosity should be returned.

If `haloLoaded=true` (which should be the default if this option is not present), then the effects of baryonic loading on the halo profile should be taken into account where necessary. Otherwise, the effects of baryonic loading should be ignored.

Galactic Component Rotation Curve

The function `Galactic_Structure_Rotation_Curve()` computes the rotation curve at a specified radius in a node. To have their contribution counted, each component must register a task using:

```
!# <rotationCurveTask>
!#  <unitName>Rotation_Curve_Procedure</unitName>
!# </rotationCurveTask>
```

where `Rotation_Curve_Procedure` is the name of a function with the following template

```
 double precision function Rotation_Curve_Procedure(thisNode,radius,componentType,massType,haloLoaded)
    type(treeNode),   intent(inout), pointer  :: thisNode
    integer,          intent(in)              :: massType,componentType
    double precision, intent(in)              :: radius
    logical         , intent(in),    optional :: haloLoaded
    .
    .
    .
    return
 end function Rotation_Curve_Procedure
```

If `componentType` is a match to the component then the procedure should return the contribution to the rotation curve due to the component matching type `massType` within `radius` for `thisNode`. `componentType` and `massType` can take one of the values described in §16.2. If `haloLoaded=true` (which should be the default if this option is not present), then the effects of baryonic loading on the halo profile should be taken into account where necessary. Otherwise, the effects of baryonic loading should be ignored.

Galactic Component Rotation Curve Gradient

The function `Galactic_Structure_Rotation_Curve_Gradient()` computes the gradient of the rotation curve at a specified radius in a node. To have their contribution counted, each component must register a task using:

```
!# <rotationCurveGradientTask>
!#  <unitName>Rotation_Curve_Gradient_Procedure</unitName>
!# </rotationCurveGradientTask>
```

where `Rotation_Curve_Gradient_Procedure` is the name of a function with the following template

```
 double precision function Rotation_Curve_Gradient_Procedure(thisNode,radius,componentType,massType,halo
    type(treeNode),   intent(inout), pointer  :: thisNode
    integer,          intent(in)              :: massType,componentType
    double precision, intent(in)              :: radius
    logical         , intent(in),    optional :: haloLoaded
    .
    .
    .
    return
 end function Rotation_Curve_Gradient_Procedure
```

If `componentType` is a match to the component then the function should return the contribution to the gradient of $V_c^2(r)$ due to the component matching type `massType` within `radius` for `thisNode`. *Note that this is the gradient of the square of the rotation curve to permit gradients to be directly summed.* `componentType` and `massType` can take one of the values described in §16.2. If `haloLoaded=true` (which should be the default if this option is not present), then the effects of baryonic loading on the halo profile should be taken into account where necessary. Otherwise, the effects of baryonic loading should be ignored.

Galactic Component Potential

The function `Galactic_Structure_Potential()` computes the potential at a specified radius in a node. To have their contribution counted, each component must register a task using:

```
 !# <potentialTask>
 !#  <unitName>Potential_Task</unitName>
 !# </potentialTask>
```

where `Potential_Task` is the name of a function with the following template

```
 double precision function Potential_Procedure(thisNode,radius,componentType,massType,haloLoaded)
    type(treeNode),   intent(inout), pointer   :: thisNode
    integer,          intent(in),    optional  :: componentType
    double precision, intent(in)                :: radius
    logical         , intent(in),    optional :: haloLoaded
    .
    .
    return
 end function Potential_Procedure
```

If `componentType` is a match to the component then the procedure should return the contribution to the rotation curve due to the component matching type `massType` within `radius` for `thisNode`. `componentType` and `massType` can take one of the values described in §16.2. If `haloLoaded=true` (which should be the default if this option is not present), then the effects of baryonic loading on the halo profile should be taken into account where necessary. Otherwise, the effects of baryonic loading should be ignored.

Galactic Component Surface Density

The function `Galactic_Structure_Surface_Density()` computes the surface density of material at a given position within a node. Note that while a 3-D position is specified the routine should return the surface density corresponding to integrating the component density through the minor axis (typically the z-axis). To have their surface density counted, each component must register a task using:

```
!# <surfaceDensityTask>
!#  <unitName>Surface_Density_Procedure</unitName>
!# </surfaceDensityTask>
```

where `Surface_Density_Procedure` is the name of a function with the following template

```
double precision function Surface_Density_Procedure(thisNode,position,coordinateSystem,componentType,ma
   type(treeNode),   intent(inout), pointer     :: thisNode
   integer,          intent(in)                 :: massType,coordinateSystem,componentType
   double precision, intent(in),    dimension(3):: position
   logical         , intent(in),    optional    :: haloLoaded
   .
   .
   .
   return
end function Surface_Density_Procedure
```

If `componentType` is a match to the component then the function should return the surface density of the component matching type `massType` at `position` for `thisNode`. `componentType` and `massType` can take one of the values described in §16.2. The coordinate system in which `position` is specified is given by `coordinateSystem` which can take on the following values:

`coordinateSystemCartesian` Cartesian (x, y, z);

`coordinateSystemSpherical` Spherical (r, θ, ϕ);

`coordinateSystemCylindrical` Cylindrical (R, ϕ, z).

If `haloLoaded`=`true` (which should be the default if this option is not present), then the effects of baryonic loading on the halo profile should be taken into account where necessary. Otherwise, the effects of baryonic loading should be ignored.

Halo Formation Events

Tasks to be performed when a halo is deemed to have "formed" (or reformed) can be registered using the `haloFormationTask` directive. For example, the `Tree_Node_Methods_Hot_Halo` module registers a task using

```
!# <haloFormationTask>
!#  <unitName>Hot_Halo_Formation_Task</unitName>
!# </haloFormationTask>
```

The contents of `<unitName>` should give the name of the subroutine to be called on halo formation. The subroutine should have a single argument, `thisNode`, which is the node that has (re)formed.

HDF5 File Close

Tasks to be performed just prior to closing the Galacticus output HDF5 file (typically involving writing accumulated data to that file) can be registered using the `hdfPreCloseTask` directive. For example, the `Merger_Tree_Timesteps_History` module registers a task using

```
!# <hdfPreCloseTask>
!#  <unitName>Merger_Tree_History_Write</unitName>
!# </hdfPreCloseTask>
```

The contents of `<unitName>` should give the name of the subroutine to be called prior to HDF5 file closure. The subroutine should have no arguments.

Merger Tree Extra Output Tasks

Extra outputs for merger trees (i.e. those which do not involve output of a fixed number of properties for every node—examples might be star formation histories for a subset of galaxies) can be added using the directive: `mergerTreeExtraOutputTask`. The directive should give the name of the subroutine to be called to perform the task. A template for this task is:

```
!# <mergerTreeExtraOutputTask>
!#  <unitName>Galacticus_Extra_Output_Example</unitName>
!# </mergerTreeExtraOutputTask>
subroutine Galacticus_Extra_Output_Example(thisNode,iOutput,treeIndex,nodePassesFilter)
  implicit none
  type(treeNode),          intent(inout), pointer :: thisNode
  integer,                 intent(in)             :: iOutput
  integer(kind=kind_int8), intent(in)             :: treeIndex
  logical,                 intent(in)             :: nodePassesFilter
  .
  .
  .
  return
end subroutine Galacticus_Extra_Output_Example
```

The subroutine will be called for each node in each merger tree at each output, and should perform whatever extra output related to `thisNode`. The index of the output and tree are provided as `iOutput` and `treeIndex` for reference, and may be used in organizing output. The `nodePassesFilter` flag will be set to `true` if `thisNode` passed all active output filters (see §16.4.1). If it is `false` then typically no output should occur (although other tasks may still be undertaken).

Merger Tree Output Tasks

Additional outputs for merger trees can be added using three directives: `mergerTreeOutputPropertyCount`, `mergerTreeOutputNames` and `mergerTreeOutputTask`. Each directive should give the name of the subroutine to be called to perform the task and, additionally, a name for sorting (this should be the same for all three directives and ensures that output tasks are always called in the correct order). Templates for these tasks are:

```
!# <mergerTreeOutputNames>
!#  <unitName>Galacticus_Output_Tree_Example_Names</unitName>
!#  <sortName>Galacticus_Output_Tree_Example</sortName>
!# </mergerTreeOutputNames>
subroutine Galacticus_Output_Tree_Example_Names(integerProperty,integerPropertyNames,integerPropertyCo
     &,doubleProperty,doublePropertyNames,doublePropertyComments,doublePropertyUnitsSI,time)
  implicit none
  double precision, intent(in)                  :: time
  integer,          intent(inout)               :: integerProperty,doubleProperty
  character(len=*), intent(inout), dimension(:) :: integerPropertyNames,integerPropertyComments,double
       &,doublePropertyComments
  double precision, intent(inout), dimension(:) :: integerPropertyUnitsSI,doublePropertyUnitsSI
  .
  .
  .
  return
```

```
  end subroutine Galacticus_Output_Tree_Example_Names

  !# <mergerTreeOutputPropertyCount>
  !#  <unitName>Galacticus_Output_Tree_Example_Property_Count</unitName>
  !#  <sortName>Galacticus_Output_Tree_Example</sortName>
  !# </mergerTreeOutputPropertyCount>
  subroutine Galacticus_Output_Tree_Example_Property_Count(integerPropertyCount,doublePropertyCount)
    implicit none
    integer, intent(inout) :: integerPropertyCount,doublePropertyCount
    .
    .
    .
    return
  end subroutine Galacticus_Output_Tree_Example_Property_Count

  !# <mergerTreeOutputTask>
  !#  <unitName>Galacticus_Output_Tree_Example</unitName>
  !#  <sortName>Galacticus_Output_Tree_Example</sortName>
  !# </mergerTreeOutputTask>
  subroutine Galacticus_Output_Tree_Example(thisNode,integerProperty,integerBufferCount,integerBuffer,do
       &,doubleBufferCount,doubleBuffer)
    implicit none
    type(treeNode),          intent(inout), pointer :: thisNode
    integer,                 intent(inout)          :: integerProperty,integerBufferCount,doubleProperty
    integer(kind=kind_int8), intent(inout)          :: integerBuffer(:,:)
    double precision,        intent(inout)          :: doubleBuffer(:,:)
    .
    .
    .
    return
  end subroutine Galacticus_Output_Tree_Example
```

The `mergerTreeOutputPropertyCount` subroutine must simply increment `integerPropertyCount` and `doublePropertyCount` by the number of integer and double precision properties that will be output respectively. The `mergerTreeOutputNames` subroutine must store the dataset names, comments and units in the SI system[8] for each integer and double precision property in the supplied arrays. The value of `integerProperty` and `doubleProperty` should be incremented by 1 before each property name/comment is set—these then supply the position within the input arrays in which to store the name. The `mergerTreeOutputTask` subroutine must similarly place the desired property values for `thisNode` into the supplied arrays. The value of `integerProperty` and `doubleProperty` should be incremented by 1 before each property value is set. The value can then be stored in, for example, `integerBuffer(integerBufferCount,integerPrope`

Merger Tree Pre-Construction Tasks

Additional tasks to be performed prior to the construction of each merger tree can be added using the `mergerTreePreTreeConstructionTask` directive. For example, the tree timing task uses this directive as follows:

```
  !# <mergerTreePreTreeConstructionTask>
```

[8] For dimensionless quantities, the units may be set to zero. In such cases, the `unitsInSI` attribute for the dataset will not be written to the Galacticus output file.

```
!#   <unitName>Meta_Tree_Timing_Pre_Construction</unitName>
!# </mergerTreePreTreeConstructionTask>
```

Here, `Meta_Tree_Timing_Pre_Construction` is the name of a subroutine which will be called to perform whatever tasks are required. The subroutine must have the following form:

```
 subroutine Merger_Tree_PreConstruction_Task()
  implicit none
  .
  .
  .
  return
end subroutine Merger_Tree_PreConstruction_Task
```

The subroutine will be called once for each tree, before the tree has been constructed.

Merger Tree Post-Evolution Tasks

Additional tasks to be performed after the evolution (and subsequent destruction) of each merger tree can be added using the `mergerTreePostEvolveTasker` directive. For example, the tree timing task uses this directive as follows:

```
!# <mergerTreePostEvolveTask>
!#   <unitName>Meta_Tree_Timing_Post_Evolve</unitName>
!# </mergerTreePostEvolveTask>
```

Here, `Meta_Tree_Timing_Post_Evolve` is the name of a subroutine which will be called to perform whatever tasks are required. The subroutine must have the following form:

```
 subroutine Merger_Tree_PostEvolution_Task()
  implicit none
  .
  .
  .
  return
end subroutine Merger_Tree_PostEvolution_Task
```

The subroutine will be called once for each tree, after the tree has been evolved and destroyed.

Merger Tree Pre-Evolution Tasks

Additional tasks to be performed on merger trees prior to their evolution can be added using the `mergerTreePreEvolveTask` directive. For example, the mass accretion history task uses this directive as follows:

```
!# <mergerTreePreEvolveTask>
!#   <unitName>Merger_Tree_Mass_Accretion_History_Output</unitName>
!# </mergerTreePreEvolveTask>
```

Here, `Merger_Tree_Mass_Accretion_History_Output` is the name of a subroutine which will be called to perform whatever tasks are required. The subroutine must have the following form:

```
 subroutine Merger_Tree_PreEvolution_Task(thisTree)
  implicit none
  type(mergerTree), intent(in) :: thisTree
  .
  .
  .
  return
end subroutine Merger_Tree_PreEvolution_Task
```

where `thisTree` is the tree to be processed. The function will be called once for each tree, prior to the tree being evolved. Note that `thisTree` may link to other trees via its `nextTree` pointer. The function may want to process each tree in this linked list.

Merger Tree Initialization Tasks

Additional tasks to be performed during merger tree initialization can be added using the `mergerTreeInitializeTask` directive. The directive should contain a single argument, giving the name of a subroutine to be called to initialize the method. For example, the `standard` basic component method uses this directive as follows:

```
!# <satelliteMergerTask>
!#  <unitName>Halo_Mass_Accretion_Rate</unitName>
!# </satelliteMergerTask>
```

Here, `Halo_Mass_Accretion_Rate` is the name of a subroutine which will be called to perform whatever tasks are required as a result of the merger. The subroutine must have the following form:

```
 subroutine Merger_Tree_Initialize_Task(thisNode)
  implicit none
  type(treeNode), pointer, intent(inout) :: thisNode
  .
  .
  .
  return
end subroutine Merger_Tree_Initialize_Task
```

where `thisNode` is the node to be initialized. The subroutine will be called once for each node in the tree.

Merger Tree Structure Output Tasks

Additional outputs for merger tree structure output can be added using the `mergerTreeStructureOutputTask`. The directive should give the name of the subroutine to be called to perform the task. The templates for this tasks is:

```
!# <mergerTreeStructureOutputTask>
!#  <unitName>Structure_Output_Task</unitName>
!# </mergerTreeStructureOutputTask>
subroutine Structure_Output_Task(baseNode,nodeProperty,treeGroup)
  implicit none
  type(treeNode),   intent(in),    pointer      :: baseNode
  double precision, intent(inout), dimension(:) :: nodeProperty
  type(hdf5Object), intent(inout)               :: treeGroup
  .
```

```
    .
    .
    return
  end subroutine Structure_Output_Task
```

The subroutine must walk the merger tree beginning from the given `baseNode` and store each property to output in the given `nodeProperty` array. Once populated, this array can be written to the appropriate HDF5 group, given by `treeGroup`, in the GALACTICUS output file.

Node Dump

The function `Node_Dump(thisNode)` writes out all properties of a node to the display. To have their properties listed, each component must register a task using:

```
!# <nodeDumpTask>
!#  <unitName>Node_Dump_Procedure</unitName>
!# </nodeDumpTask>
```

where `Node_Dump_Procedure` is the name of a subroutine with the following template

```
 subroutine Node_Dump_Procedure(thisNode)
    type(treeNode),   intent(inout), pointer :: thisNode
    .
    .
    .
    return
 end subroutine Node_Dump_Procedure
```

If the node contains an active component, this subroutine should display all relevant properties of the component. If not, it can display a short message indicating that fact.

Output Group Output Tasks

Extra outputs for output groups (i.e. the groups which hold all merger tree data for a given output time) can be added using the directive: `outputGroupOutputTask`. The directive should give the name of the subroutine to be called to perform the task. A template for this task is:

```
  !# <outputGroupOutputTask>
  !#  <unitName>Galacticus_Output_Group_Output_Example</unitName>
  !# </outputGroupOutputTask>
  subroutine Galacticus_Output_Group_Output_Example(outputGroup,time)
    implicit none
    type(hdf5Object), intent(inout) :: outputGroup
    double precision, intent(in)    :: time
    .
    .
    .
    return
  end subroutine Galacticus_Output_Group_Output_Example
```

The subroutine will be called for each output group created, and should perform whatever extra output it requires. The `outputGroup` object and the corresponding output `time` are provided as input parameters.

Post-evolve Tasks

Additional methods for post-evolve tasks (i.e. things that should be done after each node is evolved differentially to the next output time, interrupt, or event) can be added using the `postEvolveTask` directive. The directive should contain a single argument, giving the name of a subroutine to be called to perform the task. For example, the standard hot halo component adds a task as follows:

```
!# <postEvolveTask>
!#  <unitName>Tree_Node_Hot_Halo_Postevolve_Standard</unitName>
!# </postEvolveTask>
```

Here, `Tree_Node_Hot_Halo_Postevolve_Standard` is the name of a subroutine which will be called to perform whatever tasks are required. The subroutine must have the following form:

```
 subroutine Post_Evolve_Task(node)
  implicit none
  type(treeNode), intent(inout) :: node
  .
  .
  .
  return
end subroutine Post_Evolve_Task
```

where `node` is the node for which tasks will be performed. Tasks typically involve cleaning up after differential evolution.

Post-step Tasks

Additional methods for post-step tasks (i.e. things that should be done after each ODE solver step when evoling a node differentially) can be added using the `postStepTask` directive. The directive should contain a single argument, giving the name of a subroutine to be called to perform the task. For example, the standard hot halo component adds a task as follows:

```
!# <postStepTask>
!#  <unitName>Tree_Node_Hot_Halo_Poststep_Standard</unitName>
!# </postStepTask>
```

Here, `Tree_Node_Hot_Halo_Poststep_Standard` is the name of a subroutine which will be called to perform whatever tasks are required. The subroutine must have the following form:

```
 subroutine Post_Step_Task(node,status)
  implicit none
  type   (treeNode), intent(inout) :: node
  integer          , intent(inout) :: status
  .
  .
  .
  return
end subroutine Post_Step_Task
```

where `node` is the node for which tasks should be performed. If any change is made to the state of the node then `status` should be set equal to `FGSL_Failure`. Tasks typically involve cleaning up after differential evolution.

Pre-derivative Tasks

Additional methods for pre-derivative tasks (i.e. things that should be done just prior to the computation of derivatives or properties for a node) can be added using the `preDerivativeTask` directive. The directive should contain a single argument, giving the name of a subroutine to be called to perform the task. For example, the standard hot halo component adds a task as follows:

```
!# <preDerivativeTask>
!#  <unitName>Tree_Node_Hot_Halo_Prederivative_Standard</unitName>
!# </preDerivativeTask>
```

Here, `Tree_Node_Hot_Halo_Prederivative_Standard` is the name of a subroutine which will be called to perform whatever tasks are required. The subroutine must have the following form:

```
 subroutine Prederivative_Task(thisNode)
  implicit none
  type(treeNode), pointer, intent(inout) :: thisNode
  .
  .
  .
  return
end subroutine Prederivative_Task
```

where `thisNode` is the node for which derivatives will be computed. Tasks typically involve precomputing quantities that will be used in finding the derivatives.

Radius Solver Tasks

Galactic radii solver functions (see §16.4.1) need to be able to interact with the components of a tree node to

1. Determine which components want a radius to be solved for;
2. Get and set the properties of those components.

The `radiusSolverPlausibility` and `radiusSolverTask` directives facilitate this. A component which has a radius to be solved for should include directives of the form:

```
!# <radiusSolverTask>
!#  <unitName>Component_Radius_Solver_Plausibility</unitName>
!# </radiusSolverTask>
```

and

```
!# <radiusSolverTask>
!#  <unitName>Component_Radius_Solver</unitName>
!# </radiusSolverTask>
```

where `Component_Radius_Solver_Plausibility` is the name of a subroutine which will specify whether or not the component is physically plausible for radius solving (e.g. has non-negative mass) and should have the following form:

```
subroutine Component_Radius_Solver_Plausibility(thisNode,galaxyIsPhysicallyPlausible)
   implicit none
   type(treeNode), pointer, intent(inout) :: thisNode
```

```
   logical,                intent(inout) :: galaxyIsPhysicallyPlausible
   .
   .
   .
   return
end subroutine Component_Radius_Solver_Plausibility
```

which should set `galaxyIsPhysicallyPlausible` to false if the component is not physically plausible, but should otherwise leave `galaxyIsPhysicallyPlausible` unchanged. Additionally, `Component_Radius_Solver` is the name of a subroutine which will supply the necessary information about the node, and which should have the following form:

```
subroutine Component_Radius_Solver(thisNode,componentActive,specificAngularMomentum,Radius_Get,Radius_S
   implicit none
   type(treeNode),   pointer, intent(inout) :: thisNode
   logical,                   intent(out)   :: componentActive
   double precision,          intent(out)   :: specificAngularMomentum
   procedure(),      pointer, intent(out)   :: Radius_Get,Velocity_Get
   procedure(),      pointer, intent(out)   :: Radius_Set,Velocity_Set
   .
   .
   .
   return
end subroutine Component_Radius_Solver
```

When called, the subroutine should set `componentActive` to indicate whether or not this nod contains an active component of the type. If it does, it should also set `specificAngularMomentum` to reflect the specific angular momentum (in km s^{-1} Mpc) of the component (at whatever point in its profile the radius is required) and should point the four procedure pointers to routines which get and set the radius and circular velocity properties of the component (which should have the standard form for component get and set methods). It is acceptable for the set procedures to point to dummy routines.

The galactic structure radii solver routines will use this information to determine (and set) the radius and circular velocity of the component. An advantage of this approach is that different radii solver methods can all use this same system, ensuring that just a single interface is needed in each component.

Satellite Host Change Tasks

Additional methods for satellite host change events (i.e. when a satellite node moves to a new host) can be added using the `satelliteHostChangeTask` directive. The directive should contain a single argument, giving the name of a subroutine to be called to initialize the method. For example, the `simple` satellite orbits components uses this directive as follows:

```
!# <satelliteHostChangeTask>
!#  <unitName>Satellite_Orbit_New_Host</unitName>
!# </satelliteHostChangeTask>
```

Here, `Satellite_Orbit_New_Host` is the name of a subroutine which will be called to perform whatever tasks are required as a result of the host change. The subroutine must have the following form:

```
subroutine New_Host_Task(thisNode)
 implicit none
 type(treeNode), pointer, intent(inout) :: thisNode
```

```
    .
    .
    .
    return
  end subroutine New_Host_Task
```

where `thisNode` is the node which has changed host (the new host halo is `thisNode%parentNode`).

Satellite Merger Tasks

Additional methods for satellite merger tasks can be added using the `satelliteMergerTask` directive. The directive should contain a single argument, giving the name of a subroutine to be called to initialize the method. For example, the `simple` satellite orbits components uses this directive as follows:

```
!# <satelliteMergerTask>
!#  <unitName>Satellite_Merger_Task</unitName>
!# </satelliteMergerTask>
```

Here, `Satellite_Merger_Task` is the name of a subroutine which will be called to perform whatever tasks are required as a result of the merger. The subroutine must have the following form:

```
  subroutine Satellite_Merger_Task(thisNode)
   implicit none
   type(treeNode), pointer, intent(inout) :: thisNode
   .
   .
   .
   return
  end subroutine Satellite_Merger_Task
```

where `thisNode` is the node about to merge with `thisNode%parentNode`.

Star Formation History Tasks

Additional methods for star formation history tracking can be added using the `starFormationHistoriesMethod` directive. The directive should contain a single argument, giving the name of a subroutine to be called to initialize the method. For example, the `metallicitySplit` method uses this directive as follows:

```
!# <starFormationHistoriesMethod>
!#  <unitName>Star_Formation_Histories_Metallicity_Split_Initialize</unitName>
!# </starFormationHistoriesMethod>
```

Here, `Star_Formation_Histories_Metallicity_Split_Initialize` is the name of a subroutine which will be called to initialize the method. The subroutine must have the following form:

```
  subroutine Method_Initialize( starFormationHistoriesMethod     &
       &                       ,Star_Formation_History_Create_Do &
       &                       ,Star_Formation_History_Scales_Do &
       &                       ,Star_Formation_History_Record_Do &
       &                       ,Star_Formation_History_Output_Do &
       &                      )
    implicit none
    type(varying_string),          intent(in)    ::  starFormationHistoriesMethod
```

```
   procedure(),          pointer, intent(inout) ::  Star_Formation_History_Create_Do &
      &                                            ,Star_Formation_History_Scales_Do &
      &                                            ,Star_Formation_History_Record_Do &
      &                                            ,Star_Formation_History_Output_Do

   if (starFormationHistoriesMethod == 'myMethod') then
      Star_Formation_History_Create_Do => My_Create
      Star_Formation_History_Scales_Do => My_Scales
      Star_Formation_History_Record_Do => My_Record
      Star_Formation_History_Output_Do => My_Output
   end if
   return
 end subroutine Method_Initialize
```

where `myMethod` is the name of this method as will be specified by the `starFormationHistoriesMethod` input parameter. The procedure pointers must be set to point to subroutines which perform the functions described below. The initialization subroutine should perform any other tasks required to initialize the module (such as reading parameters etc.).

The `Star_Formation_History_Create_Do` subroutine must have the form:

```
 subroutine My_Create(thisNode,thisHistory)
   implicit none
   type(treeNode), intent(inout), pointer :: thisNode
   type(history),   ntent(inout)          :: thisHistory
   return
 end subroutine My_Create
```

and should return a history object in `thisHistory` suitable for holding a star formation history for `thisNode`.

The `Star_Formation_History_Scales_Do` subroutine must have the form:

```
 subroutine My_Scales(thisHistory,stellarMass,stellarAbundances)
   implicit none
   double precision,          intent(in)    :: stellarMass
   type(abundancesStructure), intent(in)    :: stellarAbundances
   type(history),             intent(inout) :: thisHistory
   return
 end subroutine My_Scales
```

and should set the ODE solver error tolerance scales in `thisHistory`, using the provided information on `stellarMass` and `stellarAbundances` if required.

The `Star_Formation_History_Record_Do` subroutine must have the form:

```
 subroutine My_Record(thisNode,thisHistory,fuelAbundances,starFormationRate)
   implicit none
   type(treeNode),            intent(inout), pointer :: thisNode
   type(history),             intent(inout)          :: thisHistory
   type(abundancesStructure), intent(in)             :: fuelAbundances
   double precision,          intent(in)             :: starFormationRate
   return
 end subroutine My_Record
```

and should record the contribution to the star formation history in `thisHistory` for `thisNode` given the current `starFormationRate` and star formation `fuelAbundances`. That is, the subroutine should adjust the rates in `thisHistory` appropriately.

The `Star_Formation_History_Output_Do` subroutine must have the form:

```
subroutine My_Output(thisNode,thisHistory,iOutput,treeIndex,componentLabel)
  implicit none
  type(treeNode),          intent(inout), pointer :: thisNode
  type(history),           intent(inout)          :: thisHistory
  integer,                 intent(in)             :: iOutput
  integer(kind=kind_int8), intent(in)             :: treeIndex
  character(len=*),        intent(in)             :: componentLabel
  return
end subroutine My_Output
```

and should write the star formation history, `thisHistory`, for `thisNode` to the output file. The output number and tree index are provided as `iOutput` and `treeIndex` for reference, and `componentLabel` provides a suitable label for the component to which the history belongs (and so should be used in the name of the datasets to which the history is written for example).

Conventionally, star formation histories are output as follows:

```
HDF5 "galacticus.hdf5" {
GROUP "starFormationHistories" {
   COMMENT "Star formation history data."
   GROUP "Output1" {
      COMMENT "Star formation histories for all trees at each out"
      GROUP "mergerTree1" {
         COMMENT "Star formation histories for each tree."
         DATASET "diskSFH<nodeID>" {
         COMMENT "Star formation history stellar masses of the disk "
            DATATYPE  H5T_IEEE_F64LE
            DATASPACE  SIMPLE { }
         }
         DATASET "diskTime<nodeID>" {
         COMMENT "Star formation history times of the disk component"
            DATATYPE  H5T_IEEE_F64LE
            DATASPACE  SIMPLE { }
         }
         DATASET "spheroidSFH<nodeID>" {
         COMMENT "Star formation history stellar masses of the spher"
            DATATYPE  H5T_IEEE_F64LE
            DATASPACE  SIMPLE { }
         }
         DATASET "spheroidTime<nodeID>" {
         COMMENT "Star formation history times of the spheroid compo"
            DATATYPE  H5T_IEEE_F64LE
            DATASPACE  SIMPLE { }
         }
      }
      GROUP "mergerTree2" {
      .
```

```
      .
      .
      }
   }
   GROUP "Output1" {
   .
   .
   .
   }
}
}
```

where `nodeID` is the index of the relevant node. The specifics of each dataset will depend on the selected star formation history method.

Currently defined star formation history methods are:

`metallicitySplit` The star formation history is tabulated on a grid of time and metallicity. The binning in time is chosen such that bins are at most of size `[starFormationHistoryTimeStep]` between the time at which each galaxy formed and the final output time, and at most of size `[starFormationHistoryFineTimeStep]` in the period `[starFormationHistoryFineTime]` prior to each output time (all times specified in Gyr). The allows fine binning of recent star formation just prior to each output. Usually, the metallicity binning is arranged logarithmically in metallicity with `[starFormationHistoryMetallicityCount]` bins between `[starFormationHistoryMetallicityMinimum]` and `[starFormationHistoryMetallicityMaximum]` (specified in Solar units). Note that the metallicity associated with each bin is the minimum metallicity for that bin (the maximum being the metallicity value associated with the next bin, except for the final bin which extends to infinite metallicity). If `[starFormationHistoryMetallicityCount]`$= 0$ is set, then the star formation history is not split by metallicity (i.e. a single metallicity bin encompassing all metallicities from zero to infinity is used). Alternatively, specific metallicity bin boundaries can be set via the `[starFormationHistoryMetallicityBoundaries]` parameter—a final boundary corresponding to infinity is always added automatically. Output follows the conventional format, with 2D star formation history datasets to represent the history as a function of time and metallicity. An additional `metallicities` dataset is added to the `starFormationHistories` output group to record the metallicity binning as follows:

```
DATASET "metallicities" {
 COMMENT "Metallicities at which star formation histories are tabulated"
   DATATYPE  H5T_IEEE_F64LE
   DATASPACE  SIMPLE { ( [starFormationHistoryMetallicityCount] ) / ( [starFormationHistoryMetallic
}
```

`inSitu` The star formation history is tabulated on a grid of time and is split between in-situ and accreted star formation. The time grid is the same as (and controlled by the same parameters) are for the `metallicitySplit` method. Output follows the conventional format, with 2D star formation history datasets to represent the history as a function of time and origin. The first element in the origin dimension records in-situ star formation, while the second element records total star formation.

16.5. Subsystems

This section describes some of the subsystems within Galacticus that support various physical entities or processes.

16.5.1. Kepler Orbits

The `keplerOrbit` object (provided by the `Kepler_Orbits_Structure` module) stores the parameters of a single Keplerian orbit. It internally handles computation of additional/alternate orbital parameters once an orbit has been fully defined. Currently, the orientation of orbits (i.e. the unit vector normal to the orbital plane and the argument of periapsis) is not tracked. As such, orbits are fully defined by three parameters (in addition to the masses of the orbitting bodies). The following limitations presently apply to the `keplerOrbit` object:

- If an orbit is overdefined (i.e. if more than three parameters are set manually) no checking is performed to ensure that the parameters are consistent with a Keplerian orbit;
- Not all interconversions between parameters are supported[9]. If a conversion cannot be performed, an error message will be given.

A `keplerOrbit` object can be reset by calling the `reset()` method, and its defined/undefined status can be tested with the `isDefined()` method or asserted with the `assertIsDefined()` method. The following orbital parameters are supported, each method returning the value of the parameter and a corresponding method suffixed with `Set` can be used to set the parameter: `radius`, `velocityRadial`, `velocityTengentail`, `energy`, `angularMomentum`, `eccentricity`, `semiMajorAxis`, `radiusPericenter`, `radiusApocenter`. Additionally, the masses of the orbitting bodies are provided by the `hostMass()` and `reducedMassSpecific()`$= M_{\rm host}/(M_{\rm host}+M_{\rm satellite})$ methods. Finally, the `velocityScale()` method returns $GM_{\rm host}/r$ where r is the radius of the orbit.

16.5.2. Chemicals

The chemicals subsystem provides both a interface to a database of known chemicals (allowing their physical properties to be queried) and a structure to store abundances/masses/etc. of the set of chemicals being tracked in Galacticus. The name "chemicals" is used to denote any chemical species that might be involved in reactions, including molecules, atoms, atomic and molecular ions and electrons.

Chemical Database

The file `data/Chemical_Database.cml` contains a database of chemicals that can currently be used by Galacticus. It uses a simplified version of the Chemical Markup Language to describe chemicals. An excerpt from the database is shown below:

```
<list>
 <chemical>
  <id>MolecularHydrogenAnion</id>
  <formalCharge>-1</formalCharge>
  <atomArray>
    <atom>
     <id>1</id>
     <elementType>H</elementType>
    </atom>
    <atom>
     <id>2</id>
     <elementType>H</elementType>
```

[9] The `keplerOrbit` object works by trying to convert to the combination radius, radial and tengential velocities. Once these are defined, all other parameters can be computed. However, for orbits defined in terms of other parameters, the `keplerOrbit` object does not know how to convert from every such combination of parameters.

```
    </atom>
   </atomArray>
   <bondArray>
    <bond>
     <atomRefs2>1 2</atomRefs2>
     <order>1</order>
    </bond>
   </bondArray>
  </chemical>
  .
  .
  .
 </list>
```

The database contains a `list` of chemicals, each contained within a `chemical` element. The `id` element provides a label for the chemical (usually a descriptive name with no white space). The `formalCharge` element gives the charge of the chemical in units of the elementary charge. The chemical is then describe by a list of atoms and bonds inside `atomArray` and `bondArray` elements respectively. The `atomArray` can contain any number of `atom` elements, which should describe each atom in the chemical giving it a unique `id` number and an `elementType`, which is the short one or two letter label for the element (e.g. H, Ni, etc.). The `bondArray` should contain a `bond` entry for each atomic bond, which itself contains a `atomRefs2` element giving the IDs of the two atoms participating in the bond and an `order` element which gives the order of the bond (e.g. "1" for a single bond).

Chemical Structure

Within Galacticus a chemical is represeted using the `chemicalStructure` type which is provided by the `Chemical_Structures` module. A `chemicalStructure` object can be assigned a particular chemical by retrieving that chemical from the database using:

```
call myChemical%retrieve("chemicalID")
```

where `chemicalID` is the ID of the chemical in the databse. Any chemical can be exported to a CML file using

```
call myChemical%export(fileName)
```

where `fileName` gives the name of the file to which to export.

Once assigned a chemical, basic properties such as mass and charge (in atomic units) can be accessed using `myChemical%mass` and `myChemical%charge` respectively. The mass is computed from the known atomic masses of the constituent atoms of the chemical.

Chemical Abundances

Within Galacticus a set of abundances (or masses, or densities...) for all chemicals being tracked, as specified by the `[chemicalsToTrack]` input parameter, is stored within a `chemicalAbundancesStructure` type, as provided by the `Chemical_Abundances_Structure` module. The structure provides interfaces for setting and retrieving the abundance of a given chemical species, to pack/unpack all chemicals to/from an array, to convert from mass-weighted to number-weigted quantities and to multiplty and divide the chemicals abundances by a given amount. Additionally, the `Chemical_Abundances_Structure` module provides functions which provide a count of the number of chemicals tracked, to look up the index of a chemical array represetation from its name, and to retrieve the name of a given chemical.

16.5.3. Radiation

This subsystem handles radiation fields, providing convenient means to communicate radiation fields from one part of the Galacticus code to another. A radiation object can hold multiple different types of radiation field (e.g. it could contain both the cosmic microwave background and an interstellar radiation field localized to a specific galaxy).

Radiation Structure

Within Galacticus radiation fields are represented by the `radiationStructure` type which is provided by the `Radiation_Structures` module. A `radiationStructure` object must first be defined using:

```
call myRadiation%define([radiationType1,radiationType2])
```

where the list of `radiationTypes` specifies what radiation components will be present in this radiation object. Currently defined radiation types are:

`CMB` The cosmic microwave background;

`Null` A null (zero radiation) component.

For example,

```
call myRadiation%define([radiationTypeCMB])
```

will define the `myRadiation` object to contain just the cosmic microwave background.

Once defined, the specific radiation field can be set using:

```
call myRadiation%set(thisNode)
```

This will cause all components to set their radiation fields using (if necessary) the properties of `thisNode`. Radiation objects can be queried using the following methods:

`temperature(radiationTypes)` Returns the temperature (in Kelvin) of the radiation object. The optional `radiationTypes` array specifies which radiation types are to be queried.

`flux(wavelength,radiationTypes)` Returns the flux (in ergs cm^2 s^{-1} Hz^{-1} ster^{-1}) of the radiation object at the given `wavelength` (specified in units of Å). The optional `radiationTypes` array specifies which radiation types are to be queried.

16.5.4. Coordinates

The `coordinate` class, provided by the `Coordinates` module provides an object describing a position in three-dimensional space. Each extension of this class (currently, `coordinateCartesian`, `coordinateCylindrical`, and `coordinateSpherical`) supply methods to convert to and from Cartesian coordinates. The assignment operator (`=`) is overloaded such that coordinate objects of any class can be assigned to any other class and conversion to the appropriate coordinate system will happen automatically. A function accepting a `class(coordinate)` object can therefore convert it to, for example, spherical coordinates simply using

```
class(coordinate         ), intent(in) :: coordinates
type (coordinateSpherical)             :: coordinatesSpherical
coordinatesSpherical=coordinates
```

and thereby allow a position to be passed to it in any coordinate system.

Each extension of the base class also provides methods to get and set the values of each component of the relevant coordinate system (see §15.8.2 for complete details).

17. Auxilliary Methods

17.1. Conditional Stellar Mass Function

Empirical conditional stellar mass functions are used by GALACTICUS in calculations of halo mass function sampling. GALACTICUS implements the following calculations of tree processing times, which can be selected via the `[conditionalStellarMassFunctionMethod]` input parameter.

17.1.1. Behroozi (2010) Method

Currently the only option, and selected using `[conditionalStellarMassFunctionMethod]`=`Behroozi2010`, this method adopts the fitting function of Behroozi et al. [2010]:

$$\langle N_\mathrm{c}(M_\star|M)\rangle \equiv \int_{M_\star}^\infty \phi_\mathrm{c}(M'_\star)\mathrm{d}\ln M'_\star = \frac{1}{2}\left[1-\mathrm{erf}\left(\frac{\log_{10} M_\star - \log_{10} f_\mathrm{SHMR}(M)}{\sqrt{2}\sigma_{\log M_\star}}\right)\right]. \tag{17.1}$$

Here, the function $f_\mathrm{SHMR}(M)$ is the solution of

$$\log_{10} M = \log_{10} M_1 + \beta \log_{10}\left(\frac{M_\star}{M_{\star,0}}\right) + \frac{(M_\star/M_{\star,0})^\delta}{1+(M_\star/M_{\star,0})^{-\gamma}} - 1/2. \tag{17.2}$$

For satellites,

$$\langle N_\mathrm{s}(M_\star|M)\rangle \equiv \int_{M_\star}^\infty \phi_\mathrm{s}(M'_\star)\mathrm{d}\ln M'_\star = \langle N_\mathrm{c}(M_\star|M)\rangle \left(\frac{f^{-1}_\mathrm{SHMR}(M_\star)}{M_\mathrm{sat}}\right)^{\alpha_\mathrm{sat}} \exp\left(-\frac{M_\mathrm{cut}}{f^{-1}_\mathrm{SHMR}(M_\star)}\right), \tag{17.3}$$

where

$$\frac{M_\mathrm{sat}}{10^{12}M_\odot} = B_\mathrm{sat}\left(\frac{f^{-1}_\mathrm{SHMR}(M_\star)}{10^{12}M_\odot}\right)^{\beta_\mathrm{sat}}, \tag{17.4}$$

and

$$\frac{M_\mathrm{cut}}{10^{12}M_\odot} = B_\mathrm{cut}\left(\frac{f^{-1}_\mathrm{SHMR}(M_\star)}{10^{12}M_\odot}\right)^{\beta_\mathrm{cut}}. \tag{17.5}$$

By default, parameter values are taken from the fit of Leauthaud et al. [2011], specifically their `SIG_MOD1` method for their z_1 sample. These default values, and the GALACTICUS input parameters which can be used to adjust them are shown in Table 17.1. This method assumes that $P_\mathrm{s}(N|M_\star,M;\delta\ln M_\star)$ is a Poisson distribution while $P_\mathrm{c}(N|M_\star,M;\delta\ln M_\star)$ has a Bernoulli distribution, with each distribution's free parameter fixed by requiring

$$\phi(M_\star;M)\delta\ln M_\star = \sum_{N=0}^\infty N P(N|M_\star,M;\delta\ln M_\star) \tag{17.6}$$

Table 17.1.: Parameters of the Behroozi et al. [2010] conditional stellar mass function model, along with their default values and the corresponding Galacticus input parameters.

Parameter	Default	Galacticus name
$\alpha_{\rm sat}$	1.0	[conditionalStellarMassFunctionBehrooziAlphaSatellite]
$\log_{10} M_1$	12.520	[conditionalStellarMassFunctionBehrooziLog10M1]
$\log_{10} M_{\star,0}$	10.916	[conditionalStellarMassFunctionBehrooziLog10Mstar0]
β	0.457	[conditionalStellarMassFunctionBehrooziBeta]
δ	0.5666	[conditionalStellarMassFunctionBehrooziDelta]
γ	1.53	[conditionalStellarMassFunctionBehrooziGamma]
$\sigma_{\log M_\star}$	0.206	[conditionalStellarMassFunctionBehrooziSigmaLogMstar]
$B_{\rm cut}$	1.47	[conditionalStellarMassFunctionBehrooziBCut]
$B_{\rm sat}$	10.62	[conditionalStellarMassFunctionBehrooziBSatellite]
$\beta_{\rm cut}$	-0.13	[conditionalStellarMassFunctionBehrooziBetaCut]
$\beta_{\rm sat}$	0.859	[conditionalStellarMassFunctionBehrooziBetaCut]

17.2. Tree Timing

Estimates of the time taken to process a merger tree are used in some halo sampling rate functions and may in future be used in load balancing algorithms. Galacticus implements the following calculations of tree processing times, which can be selected via the `[timePerTreeMethod]` input parameter.

17.2.1. File Method

Currently the only option, and selected using `[timePerTreeMethod]=file`, this method assumes that the time taken to process a tree is given by

$$\log_{10}[\tau_{\rm tree}(M)] = \sum_{i=0}^{2} C_i (\log_{10} M)^i, \tag{17.7}$$

where M is the root mass of the tree and the coefficients C_i are read from a file, the name of which is specified via the `[timePerTreeFitFileName]` parameter. This file should be an XML document with the structure:

```
<timing>
 <fit>
   <coefficient>-0.73</coefficient>
   <coefficient>-0.20</coefficient>
   <coefficient>0.03</coefficient>
 </fit>
</timing>
```

where the array of coefficients give the values C_0, C_1 and C_2.

Note that, if Galacticus is run with `[metaCollectTimingData]=true`, then it will output measures of tree processing time to the output file (see §4.3.1). The analysis script `scripts/analysis/treeTiming.pl` can be used to extract tree timing data from such an output file and output fitting coefficients in the above format. It is used as follows:

```
treeTiming.pl <modelFile> [options.....]
```

where `<modelFile>` is the name of the GALACTICUS output file to analyze. The following options can be specified:

`accumulate` If present, this argument will cause new timing data from the `<modelFile>` is be accumulated with any timing data already present in the output file (which must be specified in this case). The fit is recomputed from the totallity of the accumulated data;

`outputFile` If present, the timing data for individual trees together with the fitting coefficiencts will be output to the specified file;

`maxPoints` When accumulating trees to the output file, this paramter, if present, will limit the number of trees stored in the file to the given value. The oldest trees added to the file will be dropped first;

`plotFile` If present, a plot of the tree timing as a function of halo mass, together with the fitting function, will be output to the specified file.

Note that the output file will contain both the fitting coefficients in the format described above and, additionally, a list of tree root masses and processing times (necessary if you later want to append trees from another run to this file).

18. Source Code Documentation

file: work/build/objects.nodes.components.Inc
Description: Auto-generated file describing the hierarchy of node and component objects.
Code lines: N/A
Contained by: file accretion.Bondi_Hoyle_Lyttleton.F90
Used by: file objects.nodes.F90

18.1. Program units

file: ANN.cpp

file: Galacticus.F90
Description: Galacticus is a semi-analytic model of galaxy formation written by Andrew Benson <abenson@carnegiescience.edu>.
Code lines: 100

program: galacticus
Description: The main Galacticus program.
Code lines: 77
Contained by: file Galacticus.F90
Modules used:
events_hooks
galacticus_banner
galacticus_error
galacticus_function_classes_destroys
galacticus_output_open
iso_varying_string
mpi_utilities
tasks
functions_global_utilities
galacticus_display_verbosity
galacticus_error_wait
galacticus_hdf5
input_parameters
mpi
system_limits

file: XRay_Absorption_ISM_Wilms2000.F90
Description: Contains a program which wraps the dotbvabs function (which implements the model of Wilms et al. 2000) from XSpec to produce a table of X-ray absorption cross-sections in the ISM. This program assumes that various files from XSpec have been downloaded into the aux/XSpec folder—usually this program will be run automatically as needed by the Galacticus::ISMCrossSections module.
Code lines: 167

function: fgabnd
Description: Function to return the abundance (relative to hydrogen) of elements. Required by dotbvabs.
Code lines: 28
Contained by: file XRay_Absorption_ISM_Wilms2000.F90

Modules used: galacticus_error

subroutine: xermsg
Description: Error message function required by dotbvabs.
Code lines: 12
Contained by: file XRay_Absorption_ISM_Wilms2000.F90
Modules used: galacticus_error

program: xray_absorption_ism_wilms2000
Description: Wraps the dotbvabs function (which implements the model of Wilms et al. 2000) from XSPEC to produce a table of X-ray absorption cross-sections in the ISM. This program assumes that various files from XSPEC have been downloaded into the aux/XSpec folder—usually this program will be run automatically as needed by the Galacticus::ISMColumnDensity module.
Code lines: 83
Contained by: file XRay_Absorption_ISM_Wilms2000.F90
Modules used: atomic_cross_sections_compton, dates_and_times, hdf5, io_hdf5, numerical_constants_prefixes, numerical_constants_units, numerical_ranges

subroutine: xwrite
Description: Message display function required by dotbvabs.
Code lines: 8
Contained by: file XRay_Absorption_ISM_Wilms2000.F90

file: accretion.Bondi_Hoyle_Lyttleton.F90
Description: Contains a module which implements calculations of Bondi-Hoyle-Lyttleton accretion (see Edgar 2004).
Code lines: 69

module: bondi_hoyle_lyttleton_accretion
Description: Implements calculations of Bondi-Hoyle-Lyttleton accretion (see Edgar 2004).
Code lines: 47
Contained by: file accretion.Bondi_Hoyle_Lyttleton.F90
Used by: subroutine node_component_black_hole_-standard_mass_accretion_rate

function: bondi_hoyle_lyttleton_accretion_radius
Description: Computes the Bondi-Hoyle-Lyttleton accretion radius (in Mpc; Edgar 2004).
Code lines: 14
Contained by: module bondi_hoyle_lyttleton_accretion
Modules used: ideal_gases_thermodynamics, numerical_constants_physical

function: bondi_hoyle_lyttleton_accretion_rate
Description: Computes the Bondi-Hoyle-Lyttleton accretion rate (in $M_\odot$ Gyr^{-1}; Edgar 2004).
Code lines: 21
Contained by: module bondi_hoyle_lyttleton_accretion
Modules used: ideal_gases_thermodynamics, numerical_constants_astronomical

numerical_constants_physical

file: accretion.halo.Bertschinger.F90
Description: An implementation of accretion from the IGM onto halos using simple truncation to mimic the effects of reionization, and the Bertschinger mass to define available mass.
Code lines: 97
Modules used: dark_matter_profiles_dmo

interface: accretionhalobertschinger
Description: Constructors for the bertschinger halo accretion class.
Code lines: 4
Contained by: file accretion.halo.Bertschinger.F90

function: bertschingerconstructorinternal
Description: Internal constructor for the bertschinger halo accretion class.
Code lines: 19
Contained by: file accretion.halo.Bertschinger.F90
Modules used: atomic_data galacticus_error

function: bertschingerconstructorparameters
Description: Default constructor for the bertschinger halo accretion class.
Code lines: 11
Contained by: file accretion.halo.Bertschinger.F90
Modules used: input_parameters

subroutine: bertschingerdestructor
Description: Destructor for the bertschinger halo accretion class.
Code lines: 7
Contained by: file accretion.halo.Bertschinger.F90

function: bertschingervelocityscale
Description: Returns the velocity scale to use for node. Use the maximum circular velocity.
Code lines: 8
Contained by: file accretion.halo.Bertschinger.F90

file: accretion.halo.F90
Description: Contains a module which implements a class implementing accretion of gas from the IGM onto halos.
Code lines: 109

module: accretion_halos
Description: Implements a class implementing accretion of gas from the IGM onto halos.
Code lines: 87
Contained by: file accretion.halo.F90
Modules used: abundances_structure chemical_abundances_structure galacticus_nodes
Used by: subroutine galacticus_function_-classes_destroy subroutine galacticus_state_retrieve

subroutine galacticus_state_store
file nodes.property_extractor.hot_mode_accretion_fraction.F90
module node_component_hot_halo_cold_mode
module node_component_hot_halo_standard
file merger_trees.operators.prune_baryons.F90
function fractionaccretionhotmodeextract
subroutine node_component_hot_halo_cold_mode_rate_compute
module node_component_hot_halo_very_simple

file: accretion.halo.Naoz_Barkana_2007.F90
Description: An implementation of accretion from the IGM onto halos using filtering mass of the IGM calculated from an equation from Naoz and Barkana [2007].
Code lines: 390
Modules used: intergalactic_medium_filtering_masses

interface: accretionhalonaozbarkana2007
Description: Constructors for the naozBarkana2007 halo accretion class.
Code lines: 4
Contained by: file accretion.halo.Naoz_Barkana_2007.F90

function: naozbarkana2007accretedmass
Description: Computes the mass of baryons accreted into node.
Code lines: 20
Contained by: file accretion.halo.Naoz_Barkana_2007.F90
Modules used: galacticus_nodes

function: naozbarkana2007accretionrate
Description: Computes the baryonic accretion rate onto node.
Code lines: 36
Contained by: file accretion.halo.Naoz_Barkana_2007.F90
Modules used: galacticus_nodes

subroutine: naozbarkana2007autohook
Description: Attach to the calculation reset event.
Code lines: 8
Contained by: file accretion.halo.Naoz_Barkana_2007.F90
Modules used: events_hooks

function: naozbarkana2007branchhasbaryons
Description: Returns true if this branch can accrete any baryons.
Code lines: 24
Contained by: file accretion.halo.Naoz_Barkana_2007.F90
Modules used: galacticus_nodes merger_tree_walkers

subroutine: naozbarkana2007calculationreset
Description: Reset the accretion rate calculation.
Code lines: 10
Contained by: file accretion.halo.Naoz_Barkana_2007.F90

function: naozbarkana2007constructorinternal
Description: Internal constructor for the naozBarkana2007 halo accretion class.
Code lines: 19
Contained by: file accretion.halo.Naoz_Barkana_2007.F90
Modules used: atomic_data galacticus_error

function: naozbarkana2007constructorparameters
Description: Constructor for the naozBarkana2007 halo accretion class which takes a parameter set as input.
Code lines: 29
Contained by: file accretion.halo.Naoz_Barkana_2007.F90
Modules used: input_parameters

subroutine: naozbarkana2007destructor
Description: Destructor for the naozBarkana2007 halo accretion class.
Code lines: 9
Contained by: file accretion.halo.Naoz_Barkana_2007.F90
Modules used: events_hooks

function: naozbarkana2007failedaccretedmass
Description: Computes the mass of baryons accreted into node.
Code lines: 20
Contained by: file accretion.halo.Naoz_Barkana_2007.F90
Modules used: galacticus_nodes

function: naozbarkana2007failedaccretionrate
Description: Computes the baryonic accretion rate onto node.
Code lines: 36
Contained by: file accretion.halo.Naoz_Barkana_2007.F90
Modules used: accretion_halo_totals galacticus_nodes

function: naozbarkana2007filteredfraction
Description: Returns the baryonic mass fraction in a halo after the effects of the filtering mass.
Code lines: 25
Contained by: file accretion.halo.Naoz_Barkana_2007.F90
Modules used: dark_matter_profile_mass_definitions galacticus_nodes

function: naozbarkana2007filteredfractioncompute
Description: Compute the filtered fraction.
Code lines: 9
Contained by: file accretion.halo.Naoz_Barkana_2007.F90

function: naozbarkana2007filteredfractionrate
Description: Returns the baryonic mass accretion rate fraction in a halo after the effects of the filtering mass.
Code lines: 31
Contained by: file accretion.halo.Naoz_Barkana_2007.F90
Modules used: dark_matter_profile_mass_definitions galacticus_nodes math_exponentiation

file: `accretion.halo.cold_mode.F90`
Description: An implementation of accretion from the IGM onto halos using simple truncation to mimic the effects of reionization and accounting for cold mode accretion.
Code lines: 406
Modules used: `cooling_functions` `kind_numbers`

interface: `accretionhalocoldmode`
Description: Constructors for the `coldMode` halo accretion class.
Code lines: 4
Contained by: file `accretion.halo.cold_mode.F90`

function: `coldmodeaccretedmass`
Description: Computes the mass of baryons accreted into `node`.
Code lines: 9
Contained by: file `accretion.halo.cold_mode.F90`

function: `coldmodeaccretedmasschemicals`
Description: Computes the mass of chemicals accreted (in $M_\odot$) onto `node` from the intergalactic medium.
Code lines: 19
Contained by: file `accretion.halo.cold_mode.F90`
Modules used: `chemical_abundances_structure`

function: `coldmodeaccretedmassmetals`
Description: Computes the mass of abundances accreted (in $M_\odot$) onto `node` from the intergalactic medium.
Code lines: 10
Contained by: file `accretion.halo.cold_mode.F90`

function: `coldmodeaccretionrate`
Description: Computes the baryonic accretion rate onto `node`.
Code lines: 9
Contained by: file `accretion.halo.cold_mode.F90`

function: `coldmodeaccretionratechemicals`
Description: Computes the rate of mass of chemicals accretion (in $M_\odot$/Gyr) onto `node` from the intergalactic medium. Assumes a primordial mixture of hydrogen and helium and that accreted material is in collisional ionization equilibrium at the virial temperature.
Code lines: 21
Contained by: file `accretion.halo.cold_mode.F90`
Modules used: `chemical_abundances_structure`

function: `coldmodeaccretionratemetals`
Description: Computes the rate of mass of abundance accretion (in $M_\odot$/Gyr) onto `node` from the intergalactic medium.
Code lines: 10
Contained by: file `accretion.halo.cold_mode.F90`

subroutine: `coldmodeautohook`
Description: Attach to the calculation reset event.
Code lines: 8

Contained by: file accretion.halo.cold_mode.F90
Modules used: events_hooks

subroutine: coldmodecalculationreset
Description: Reset the accretion rate calculation.
Code lines: 9
Contained by: file accretion.halo.cold_mode.F90

function: coldmodechemicalmasses
Description: Compute the masses of chemicals accreted (in $M_\odot$) onto node from the intergalactic medium.
Code lines: 47
Contained by: file accretion.halo.cold_mode.F90
Modules used: abundances_structure, chemical_abundances_structure, chemical_reaction_rates_utilities, galacticus_nodes, numerical_constants_astronomical

function: coldmodecoldmodefraction
Description: Computes the fraction of accretion occuring in the specified mode.
Code lines: 75
Contained by: file accretion.halo.cold_mode.F90
Modules used: abundances_structure, chemical_abundances_structure, chemical_reaction_rates_utilities, galacticus_error, galacticus_nodes, numerical_constants_astronomical, numerical_constants_atomic, numerical_constants_math, numerical_constants_physical, numerical_constants_prefixes, shocks_1d

function: coldmodeconstructorinternal
Description: Internal constructor for the coldMode halo accretion class.
Code lines: 19
Contained by: file accretion.halo.cold_mode.F90

function: coldmodeconstructorparameters
Description: Default constructor for the coldMode halo accretion class.
Code lines: 33
Contained by: file accretion.halo.cold_mode.F90
Modules used: input_parameters

subroutine: coldmodedestructor
Description: Destructor for the coldMode halo accretion class.
Code lines: 9
Contained by: file accretion.halo.cold_mode.F90
Modules used: events_hooks

function: coldmodefailedaccretedmass
Description: Computes the mass of baryons accreted into node.
Code lines: 9
Contained by: file accretion.halo.cold_mode.F90

function: `coldmodefailedaccretionrate`
Description: Computes the baryonic accretion rate onto `node`.
Code lines: 9
Contained by: file `accretion.halo.cold_mode.F90`

file: `accretion.halo.simple.F90`
Description: An implementation of accretion from the IGM onto halos using simple truncation to mimic the effects of reionization.
Code lines: 475
Modules used: `accretion_halo_totals` `chemical_states` `cosmology_functions` `cosmology_parameters` `dark_matter_halo_scales` `intergalactic_medium_state` `radiation_fields`

interface: `accretionhalosimple`
Description: Constructors for the `simple` halo accretion class.
Code lines: 4
Contained by: file `accretion.halo.simple.F90`

function: `simpleaccretedmass`
Description: Computes the mass of baryons accreted into `node`.
Code lines: 18
Contained by: file `accretion.halo.simple.F90`
Modules used: `galacticus_nodes`

function: `simpleaccretedmasschemicals`
Description: Computes the mass of chemicals accreted (in $M_\odot$) onto `node` from the intergalactic medium.
Code lines: 20
Contained by: file `accretion.halo.simple.F90`
Modules used: `chemical_abundances_structure`

function: `simpleaccretedmassmetals`
Description: Computes the mass of abundances accreted (in $M_\odot$) onto `node` from the intergalactic medium.
Code lines: 13
Contained by: file `accretion.halo.simple.F90`
Modules used: `abundances_structure`

function: `simpleaccretionrate`
Description: Computes the baryonic accretion rate onto `node`.
Code lines: 35
Contained by: file `accretion.halo.simple.F90`
Modules used: `galacticus_nodes`

function: `simpleaccretionratechemicals`
Description: Computes the rate of mass of chemicals accretion (in $M_\odot$/Gyr) onto `node` from the intergalactic medium. Assumes a primordial mixture of hydrogen and helium and that accreted material is in collisional ionization equilibrium at the virial temperature.
Code lines: 21
Contained by: file `accretion.halo.simple.F90`

Modules used: chemical_abundances_structure

function: simpleaccretionratemetals
Description: Computes the rate of mass of abundance accretion (in $M_\odot$/Gyr) onto node from the intergalactic medium.
Code lines: 13
Contained by: file accretion.halo.simple.F90
Modules used: abundances_structure

function: simplebranchhasbaryons
Description: Returns true if this branch can accrete any baryons.
Code lines: 18
Contained by: file accretion.halo.simple.F90
Modules used: merger_tree_walkers

function: simplechemicalmasses
Description: Compute the masses of chemicals accreted (in $M_\odot$) onto node from the intergalactic medium.
Code lines: 32
Contained by: file accretion.halo.simple.F90
Modules used: abundances_structure, chemical_abundances_structure, chemical_reaction_rates_utilities, galacticus_nodes, numerical_constants_astronomical

function: simpleconstructorinternal
Description: Internal constructor for the simple halo accretion class.
Code lines: 23
Contained by: file accretion.halo.simple.F90
Modules used: chemical_abundances_structure, galacticus_error, galacticus_nodes

function: simpleconstructorparameters
Description: Default constructor for the simple halo accretion class.
Code lines: 77
Contained by: file accretion.halo.simple.F90
Modules used: galacticus_error, input_parameters

subroutine: simpledestructor
Description: Destructor for the simple halo accretion class.
Code lines: 13
Contained by: file accretion.halo.simple.F90

function: simplefailedaccretedmass
Description: Computes the mass of baryons accreted into node.
Code lines: 18
Contained by: file accretion.halo.simple.F90
Modules used: galacticus_nodes

function: simplefailedaccretionrate
Description: Computes the baryonic accretion rate onto node.

Code lines: 32
Contained by: file accretion.halo.simple.F90
Modules used: galacticus_nodes

function: simplefailedfraction
Description: Returns the fraction of potential accretion onto a halo from the IGM which fails.
Code lines: 15
Contained by: file accretion.halo.simple.F90
Modules used: galacticus_nodes

function: simplevelocityscale
Description: Returns the velocity scale to use for node. Use the virial velocity.
Code lines: 8
Contained by: file accretion.halo.simple.F90

file: accretion.halo.total.Bertschinger.F90
Description: An implementation of the intergalactic medium state class for a simplistic model of instantaneous and full reionization.
Code lines: 78

interface: accretionhalototalbertschinger
Description: Constructors for the bertschinger total halo accretion class.
Code lines: 3
Contained by: file accretion.halo.total.Bertschinger.F90

function: bertschingeraccretedmass
Description: Return the mass accreted onto a halo.
Code lines: 12
Contained by: file accretion.halo.total.Bertschinger.F90
Modules used: galacticus_nodes

function: bertschingeraccretionrate
Description: Return the accretion rate onto a halo.
Code lines: 12
Contained by: file accretion.halo.total.Bertschinger.F90
Modules used: galacticus_nodes

function: bertschingerconstructorparameters
Description: Constructor for the bertschinger total halo accretion state class which takes a parameter set as input.
Code lines: 10
Contained by: file accretion.halo.total.Bertschinger.F90
Modules used: input_parameters

file: accretion.halo.total.F90
Description: Contains a module which provides a class for calculations of the intergalactic medium thermal and ionization state.
Code lines: 47

module: accretion_halo_totals

Description: Provides a class for calculations of the total accretion rate onto halos for use by the halo accretion classes which compute the accretion rates of baryonic material.
Code lines: 25
Contained by: file accretion.halo.total.F90
Modules used: galacticus_nodes
Used by: function naozbarkana2007failedaccretionrate; file accretion.halo.simple.F90; subroutine galacticus_function_classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store

file: accretion.halo.total.simple.F90
Description: An implementation of the intergalactic medium state class for a simplistic model of instantaneous and full reionization.
Code lines: 78

interface: accretionhalototalsimple
Description: Constructors for the simple total halo accretion class.
Code lines: 3
Contained by: file accretion.halo.total.simple.F90

function: simpleaccretedmass
Description: Return the mass accreted onto a halo.
Code lines: 12
Contained by: file accretion.halo.total.simple.F90
Modules used: galacticus_nodes

function: simpleaccretionrate
Description: Return the accretion rate onto a halo.
Code lines: 12
Contained by: file accretion.halo.total.simple.F90
Modules used: galacticus_nodes

function: simpleconstructorparameters
Description: Constructor for the simple total halo accretion state class which takes a parameter set as input.
Code lines: 10
Contained by: file accretion.halo.total.simple.F90
Modules used: input_parameters

file: accretion.halo.zero.F90
Description: An implementation of zero accretion from the IGM onto halos.
Code lines: 172

interface: accretionhalozero
Description: Constructors for the zero halo accretion class.
Code lines: 3
Contained by: file accretion.halo.zero.F90

function: `zeroaccretedmass`
Description: Computes the mass of baryons accreted into `node`.
Code lines: 10
Contained by: file `accretion.halo.zero.F90`

function: `zeroaccretedmasschemicals`
Description: Computes the mass of chemicals accreted (in $M_\odot$) onto `node` from the intergalactic medium.
Code lines: 12
Contained by: file `accretion.halo.zero.F90`
Modules used: `chemical_abundances_structure`

function: `zeroaccretedmassmetals`
Description: Computes the mass of abundances accreted (in $M_\odot$) onto `node` from the intergalactic medium.
Code lines: 12
Contained by: file `accretion.halo.zero.F90`
Modules used: `abundances_structure`

function: `zeroaccretionrate`
Description: Computes the baryonic accretion rate onto `node`.
Code lines: 10
Contained by: file `accretion.halo.zero.F90`

function: `zeroaccretionratechemicals`
Description: Computes the rate of mass of chemicals accretion (in $M_\odot$/Gyr) onto `node` from the intergalactic medium.
Code lines: 12
Contained by: file `accretion.halo.zero.F90`
Modules used: `chemical_abundances_structure`

function: `zeroaccretionratemetals`
Description: Computes the rate of mass of abundance accretion (in $M_\odot$/Gyr) onto `node` from the intergalactic medium.
Code lines: 12
Contained by: file `accretion.halo.zero.F90`
Modules used: `abundances_structure`

function: `zerobranchhasbaryons`
Description: Returns true if this branch can accrete any baryons.
Code lines: 9
Contained by: file `accretion.halo.zero.F90`

function: `zeroconstructorparameters`
Description: Constructor for the `zero` halo accretion class which takes a parameter set as input.
Code lines: 10
Contained by: file `accretion.halo.zero.F90`
Modules used: `input_parameters`

function: `zerofailedaccretedmass`

Description: Computes the mass of baryons accreted into `node`.
Code lines: 10
Contained by: file accretion.halo.zero.F90

function: `zerofailedaccretionrate`
Description: Computes the baryonic accretion rate onto `node`.
Code lines: 10
Contained by: file accretion.halo.zero.F90

file: `accretion_disks.ADAF.F90`
Description: Implementation of an ADAF accretion disk.
Code lines: 833
Modules used: tables

interface: `accretiondisksadaf`
Description: Constructors for the ADAF accretion disk class.
Code lines: 4
Contained by: file accretion_disks.ADAF.F90

function: `adafadiabaticindexdefault`
Description: Returns the default adiabatic index in an ADAF give the field enhancement option.
Code lines: 16
Contained by: file accretion_disks.ADAF.F90
Modules used: galacticus_error

function: `adafangularmomentum`
Description: Returns the specific angular momentum of accreted material in the ADAF.
Code lines: 14
Contained by: file accretion_disks.ADAF.F90

function: `adafconstructorinternal`
Description: Internal constructor for the ADAF accretion disk class.
Code lines: 98
Contained by: file accretion_disks.ADAF.F90
Modules used: black_hole_fundamentals, galacticus_error, numerical_constants_physical, table_labels

function: `adafconstructorparameters`
Description: Constructor for the ADAF accretion disk class which takes a parameter set as input.
Code lines: 84
Contained by: file accretion_disks.ADAF.F90
Modules used: input_parameters

subroutine: `adafdestructor`
Description: Destructor for the ADAF accretion disk class.
Code lines: 7
Contained by: file accretion_disks.ADAF.F90

function: `adafefficiencyradiative`
Description: Computes the radiative efficiency for an ADAF.

Code lines: 18
Contained by: file accretion_disks.ADAF.F90
Modules used: galacticus_error

function: adafenthalpy
Description: Returns the relativistic enthalpy of the ADAF.
Code lines: 14
Contained by: file accretion_disks.ADAF.F90

function: adafenthalpyangularmomentumproduct
Description: Return the product of enthalpy and angular momentum for the ADAF.
Code lines: 25
Contained by: file accretion_disks.ADAF.F90
Modules used: black_hole_fundamentals

function: adaffieldenhancement
Description: Returns the field enhancement factor, g, in the ADAF.
Code lines: 24
Contained by: file accretion_disks.ADAF.F90
Modules used: black_hole_fundamentals

function: adaffluidangularvelocity
Description: Returns the angular velocity of the rotating fluid with respect to the local inertial observer (ZAMO).
Code lines: 15
Contained by: file accretion_disks.ADAF.F90
Modules used: black_hole_fundamentals

function: adafgamma
Description: Returns the net relativistic boost factor from the fluid frame of an ADAF to an observer at rest at infinity. The input quantities are in natural units.
Code lines: 15
Contained by: file accretion_disks.ADAF.F90

function: adafgammaazimuthal
Description: Returns the ϕ component relativistic boost factor from the fluid frame of an ADAF to an observer at rest at infinity. The input quantities are in natural units.
Code lines: 16
Contained by: file accretion_disks.ADAF.F90
Modules used: black_hole_fundamentals

function: adafgammaradial
Description: Returns the r component relativistic boost factor from the fluid frame of an ADAF to an observer at rest at infinity. The input quantities are in natural units.
Code lines: 15
Contained by: file accretion_disks.ADAF.F90

function: adafheight
Description: Return the (dimensionless) height in an ADAF at given `radius`, for a black hole of given `spinBlackHole`.

Code lines: 17
Contained by: file accretion_disks.ADAF.F90
Modules used: black_hole_fundamentals

function: adafjetpowerblackhole
Description: Returns the power of the black hole-launched jet from an ADAF.
Code lines: 23
Contained by: file accretion_disks.ADAF.F90
Modules used: black_hole_fundamentals

function: adafjetpowerdisk
Description: Returns the power of the disk-launched jet from an ADAF.
Code lines: 18
Contained by: file accretion_disks.ADAF.F90
Modules used: black_hole_fundamentals

function: adafjetpowerdiskfromblackhole
Description: Returns the power extracted from the black hole by the disk-launched jet from an ADAF.
Code lines: 14
Contained by: file accretion_disks.ADAF.F90

function: adafpowerjet
Description: Computes the jet power of the given black hole in due to accretion from an ADAF disk.
Code lines: 19
Contained by: file accretion_disks.ADAF.F90
Modules used: black_hole_fundamentals numerical_constants_physical

function: adafratespinup
Description: Computes the spin up rate of the black hole in `thisBlackHole` due to accretion from an ADAF. disk.
Code lines: 20
Contained by: file accretion_disks.ADAF.F90

function: adaftemperature
Description: Return the dimensionless temperature of the ADAF
Code lines: 23
Contained by: file accretion_disks.ADAF.F90
Modules used: black_hole_fundamentals

function: adafvelocity
Description: Return the (dimensionless) velocity in an ADAF at given `radius`, for a black hole of given `spinBlackHole`.
Code lines: 29
Contained by: file accretion_disks.ADAF.F90
Modules used: black_hole_fundamentals

function: adafviscosityparameter
Description: Returns the effective value of the α viscosity parameter for an ADAF.
Code lines: 28

Contained by: file accretion_disks.ADAF.F90

file: accretion_disks.Eddington_limited.F90
Description: Implementation of an Eddington-limited accretion disk.
Code lines: 128

interface: accretiondiskseddingtonlimited
Description: Constructors for the Eddington-limited accretion disk class.
Code lines: 4
Contained by: file accretion_disks.Eddington_limited.F90

function: eddingtonlimitedconstructorinternal
Description: Internal constructor for the Eddington-limited accretion disk class.
Code lines: 8
Contained by: file accretion_disks.Eddington_limited.F90

function: eddingtonlimitedconstructorparameters
Description: Constructor for the Eddington-limited accretion disk class which takes a parameter set as input.
Code lines: 28
Contained by: file accretion_disks.Eddington_limited.F90
Modules used: galacticus_error input_parameters

function: eddingtonlimitedefficiencyradiative
Description: Return the radiative efficiency of an Eddington-limited accretion disk.
Code lines: 10
Contained by: file accretion_disks.Eddington_limited.F90

function: eddingtonlimitedpowerjet
Description: Return the jet power of an Eddington-limited accretion disk.
Code lines: 20
Contained by: file accretion_disks.Eddington_limited.F90
Modules used: black_hole_fundamentals numerical_constants_physical

function: eddingtonlimitedratespinup
Description: Computes the spin up rate of the black hole in `thisNode` due to accretion from an Eddington-limited accretion disk. disk. This is always zero, as no physical model is specified for this accretion disk method.
Code lines: 11
Contained by: file accretion_disks.Eddington_limited.F90

file: accretion_disks.F90
Description: Contains a module which provides a class that implements accretion disks.
Code lines: 55

module: accretion_disks
Description: Provides a class that implements accretion disks.
Code lines: 33
Contained by: file accretion_disks.F90

Modules used: galacticus_nodes
Used by: subroutine galacticus_function_classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; module node_component_black_hole_standard; program test_accretion_disks

file: accretion_disks.Shakura_Sunyaev.F90
Description: Implementation of a Shakura and Sunyaev [1973] accretion disk.
Code lines: 122

interface: accretiondiskssshakurasunyaev
Description: Constructors for the Eddington-limited accretion disk class.
Code lines: 3
Contained by: file accretion_disks.Shakura_Sunyaev.F90

function: shakurasunyaevconstructorparameters
Description: Constructor for the Shakura and Sunyaev [1973] accretion disk class which takes a parameter set as input.
Code lines: 10
Contained by: file accretion_disks.Shakura_Sunyaev.F90
Modules used: input_parameters

function: shakurasunyaevefficiencyradiative
Description: Return the radiative efficiency of a Shakura and Sunyaev [1973] accretion disk.
Code lines: 11
Contained by: file accretion_disks.Shakura_Sunyaev.F90
Modules used: black_hole_fundamentals

function: shakurasunyaevpowerjet
Description: Computes the jet power for a Shakura and Sunyaev [1973] (thin) accretion disk, using the expressions from Meier (2001; his equations 4 and 5).
Code lines: 41
Contained by: file accretion_disks.Shakura_Sunyaev.F90
Modules used: black_hole_fundamentals; numerical_constants_astronomical; numerical_constants_physical

function: shakurasunyaevratespinup
Description: Compute the rate of spin up of a black hole by a Shakura and Sunyaev [1973] accretion disk.
Code lines: 13
Contained by: file accretion_disks.Shakura_Sunyaev.F90
Modules used: black_hole_fundamentals

file: accretion_disks.spectra.F90
Description: Contains a module that implements calculations of accretion disk spectra.
Code lines: 41

module: accretion_disk_spectra
Description: Implements calculations of accretion disk spectra.

Code lines: 19
Contained by: file accretion_disks.spectra.F90
Modules used: galacticus_nodes
Used by: subroutine galacticus_function_-classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; file radiation.intergalactic_-background.internal.F90; subroutine agnspectrahopkins2008buildfileperform

file: accretion_disks.spectra.Hopkins2007.F90
Description: An implementation of the accretion disk spectra class using the model of Hopkins et al. [2007].
Code lines: 203
Modules used: file_utilities

interface: accretiondiskspectrahopkins2007
Description: Constructors for the hopkins2007 accretion disk spectra class.
Code lines: 4
Contained by: file accretion_disks.spectra.Hopkins2007.F90

subroutine: hopkins2007buildfile
Description: Build a file containing a tabulation of the Hopkins et al. [2007] model AGN spectra.
Code lines: 117
Contained by: file accretion_disks.spectra.Hopkins2007.F90
Modules used: dates_and_times; galacticus_display; galacticus_error; galacticus_paths; io_hdf5; iso_fortran_env; numerical_constants_astronomical; numerical_constants_units; numerical_ranges; string_handling; system_command

function: hopkins2007constructorinternal
Description: Constructor for the hopkins2007 accretion disk spectra class.
Code lines: 20
Contained by: file accretion_disks.spectra.Hopkins2007.F90
Modules used: galacticus_paths

function: hopkins2007constructorparameters
Description: Constructor for the hopkins2007 accretion disk spectra class.
Code lines: 10
Contained by: file accretion_disks.spectra.Hopkins2007.F90
Modules used: input_parameters

file: accretion_disks.spectra.file.F90
Description: An implementation of the accretion disk spectra class for tabulated spectra read from file.
Code lines: 188
Modules used: fgsl

interface: `accretiondiskspectrafile`
Description: Constructors for the `file` accretion disk spectra class.
Code lines: 4
Contained by: file accretion_disks.spectra.file.F90

function: `fileconstructorinternal`
Description: Internal constructor for the `file` accretion disk spectra class.
Code lines: 18
Contained by: file accretion_disks.spectra.file.F90
Modules used: array_utilities galacticus_error galacticus_nodes

function: `fileconstructorparameters`
Description: Constructor for the `file` accretion disk spectra class which takes a parameter set as input.
Code lines: 17
Contained by: file accretion_disks.spectra.file.F90

subroutine: `filedestructor`
Description: Default destructor for the `file` accretion disk spectra class.
Code lines: 17
Contained by: file accretion_disks.spectra.file.F90
Modules used: memory_management numerical_interpolation

subroutine: `fileloadfile`
Description: Load a file of AGN spectra.
Code lines: 26
Contained by: file accretion_disks.spectra.file.F90
Modules used: galacticus_error io_hdf5

function: `filespectrum`
Description: Return the accretion disk spectrum for tabulated spectra.
Code lines: 39
Contained by: file accretion_disks.spectra.file.F90
Modules used: galacticus_nodes iso_c_binding numerical_constants_astronomical numerical_constants_physical numerical_interpolation

file: `accretion_disks.switched.F90`
Description: Implementation of a switched (ADAF/thin) accretion disk.
Code lines: 267

interface: `accretiondisksswitched`
Description: Constructors for the switched accretion disk class.
Code lines: 4
Contained by: file accretion_disks.switched.F90

function: `switchedconstructorinternal`
Description: Internal constructor for the switched accretion disk class.

Code lines: 14
Contained by: file accretion_disks.switched.F90

function: switchedconstructorparameters
Description: Constructor for the switched accretion disk class which takes a parameter set as input.
Code lines: 67
Contained by: file accretion_disks.switched.F90
Modules used: galacticus_error input_parameters

subroutine: switcheddestructor
Description: Destructor for the switched accretion disk class.
Code lines: 8
Contained by: file accretion_disks.switched.F90

function: switchedefficiencyradiative
Description: Return the radiative efficiency of a switched (ADAF/thin) accretion disk.
Code lines: 14
Contained by: file accretion_disks.switched.F90

function: switchedefficiencyradiativescalingadaf
Description: Determine the scaling of radiative efficiency of the ADAF component in a switched accretion disk.
Code lines: 27
Contained by: file accretion_disks.switched.F90
Modules used: black_hole_fundamentals

function: switchedfractionadaf
Description: Decide which type of accretion disk to use.
Code lines: 33
Contained by: file accretion_disks.switched.F90
Modules used: black_hole_fundamentals

function: switchedpowerjet
Description: Return the jet power of a switched (ADAF/thin) accretion disk.
Code lines: 11
Contained by: file accretion_disks.switched.F90

function: switchedratespinup
Description: Computes the spin up rate of the black hole in thisNode due to accretion from a switched (ADAF/thin) accretion disk.
Code lines: 12
Contained by: file accretion_disks.switched.F90

file: ann_config.cpp

file: atomic.cross_sections.Compton.F90
Description: Contains a module which implements calculation of the Compton cross-section.
Code lines: 45

module: `atomic_cross_sections_compton`
Description: Implements calculation of the Compton cross-section.
Code lines: 23
Contained by: file `atomic.cross_sections.Compton.F90`
Used by: program `xray_absorption_ism_wilms2000`

function: `atomic_cross_section_compton`
Description: Returns the Compton cross section (in cm^2) for the specified `photonEnergy` (in keV) from Klein and Nishina [1929].
Code lines: 13
Contained by: module `atomic_cross_sections_compton`
Modules used: `numerical_constants_physical` `numerical_constants_prefixes` `numerical_constants_units`

file: `atomic.cross_sections.ionization.photo.F90`
Description: Contains a module that implements the atomic photo-ionization cross-section class.
Code lines: 40

module: `atomic_cross_sections_ionization_photo`
Description: Implements the atomic photo-ionization cross-section class.
Code lines: 18
Contained by: file `atomic.cross_sections.ionization.photo.F90`
Used by: file `chemical.reaction_-rates.hydrogen.F90` subroutine `galacticus_function_-classes_destroy` subroutine `galacticus_state_retrieve` subroutine `galacticus_state_store` file `radiation.intergalactic_-background.internal.F90` file `universe.operators.intergalactic_-medium_state_evolve.F90`

file: `atomic.cross_sections.ionization.photo.Verner.F90`
Description: Implementation of an atomic photoionization cross section class based on (`phfit2.f`) written by D. A. Verner (Version 2. March 25, 1996).
Code lines: 2024

interface: `atomiccrosssectionionizationphotoverner`
Description: Constructors for the `verner` atomic photoionization cross-section class.
Code lines: 3
Contained by: file `atomic.cross_sections.ionization.photo.Verner.F90`

function: `vernerconstructorparameters`
Description: Constructor for the `verner` atomic photoionization cross-section class which builds the object from a parameter set.
Code lines: 10
Contained by: file `atomic.cross_sections.ionization.photo.Verner.F90`
Modules used: `input_parameters`

function: `vernercrosssection`

Description: Computes the cross section for photo-ionization (in units of cm^2) at the specified `wavelength` for all ionization stages of all atoms from H to Zn ($Z = 30$) by use of the following fit parameters:

- Outer shells of the Opacity Project (OP) elements: Verner et al. [1996]
- Inner shells of all elements, and outer shells of the non-OP elements: Verner and Yakovlev [1995]

Original version (`phfit2.f`) written by D. A. Verner (Version 2. March 25, 1996). Inner-shell ionization energies of some low-ionized species are slightly improved to fit smoothly the experimental inner-shell ionization energies of neutral atoms.

Code lines: 73

Contained by: file atomic.cross_sections.ionization.photo.Verner.F90
Modules used: numerical_constants_physical numerical_constants_units

file: atomic.data.F90
Description: Contains a module which provides various atomic data.
Code lines: 351

module: atomic_data
Description: Provides various atomic data.
Code lines: 329
Contained by: file atomic.data.F90
Used by:

function bertschingerconstructorinternal	function naozbarkana2007constructorinternal
function massmetallicityandrews2013constructorinternal	function massmetallicityblanc2017constructorinternal
subroutine abundances_initialize	subroutine abundances_set_metallicity
function fileconstructorinternal	subroutine fileread
function nagashima2005constructorinternal	function instantaneousconstructorinternal

function: abundance_pattern_lookup
Description: Returns the position in the atoms() array of an element specified by atomic number, name or short label.
Code lines: 34
Contained by: module atomic_data
Modules used: galacticus_error string_handling

function: atom_lookup
Description: Returns the position in the atoms() array of an element specified by atomic number, name or short label.
Code lines: 44
Contained by: module atomic_data
Modules used: galacticus_error string_handling

function: atomic_abundance
Description: Returns the abundance by mass of a given atom in a given abundance pattern.
Code lines: 32
Contained by: module atomic_data

function: atomic_data_atoms_count
Description: Return the number of atomic species known in this module.
Code lines: 10
Contained by: module atomic_data

subroutine: atomic_data_initialize
Description: Ensure that the module is initialized by reading in data.
Code lines: 117
Contained by: module atomic_data
Modules used: fox_dom galacticus_error

galacticus_paths
iso_varying_string
string_handling
io_xml
memory_management

function: `atomic_mass`
Description: Returns the atomic mass of an element specified by atomic number, name or short label.
Code lines: 21
Contained by: module atomic_data

function: `atomic_short_label`
Description: Return the short label for an atom.
Code lines: 19
Contained by: module atomic_data

type: `atomicdata`
Description: Data type for storing atomic data.
Code lines: 7
Contained by: module atomic_data

file: `atomic.ionization_potentials.F90`
Description: Contains a module that implements an atomic ionization potential class.
Code lines: 41

module: `atomic_ionization_potentials`
Description: Implements an atomic ionization potential class.
Code lines: 17
Contained by: file atomic.ionization_potentials.F90
Used by: file atomic.radiation.gaunt_-factors.Sutherland1998.F90
subroutine galacticus_function_-classes_destroy
subroutine galacticus_state_store
file universe.operators.intergalactic_-medium_state_evolve.F90
file atomic.radiation.gaunt_-factors.vanHoof2014.F90
subroutine galacticus_state_retrieve
program test_gaunt_factors

file: `atomic.ionization_potentials.Verner.F90`
Description: Implements an atomic ionization potential class, which provides potentials for all ionization stages of all atoms from H to Zn using data taken from Dima Verner's code.
Code lines: 542

interface: `atomicionizationpotentialverner`
Description: Constructors for the `verner` atomic ionization potential class.
Code lines: 3
Contained by: file atomic.ionization_potentials.Verner.F90

function: `vernerconstructorparameters`
Description: Constructor for the `verner` atomic ionization potentail class which builds the object from a parameter set.
Code lines: 10
Contained by: file atomic.ionization_potentials.Verner.F90

Modules used: input_parameters

function: vernerpotential
Description: Return the ionization potential (in units of electon volts) for the ion with given atomicNumber and electronNumber using data taken from Dima Verner's code.
Code lines: 11
Contained by: file atomic.ionization_potentials.Verner.F90

file: atomic.radiation.gaunt_factors.F90
Description: Contains a module which provides a class implenting Gaunt factors.
Code lines: 40

module: atomic_radiation_gaunt_factors
Description: Provides a class implenting Gaunt factors.
Code lines: 18
Contained by: file atomic.radiation.gaunt_factors.F90
Used by: subroutine galacticus_function_classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; program test_gaunt_factors; file universe.operators.intergalactic_medium_state_evolve.F90

file: atomic.radiation.gaunt_factors.Sutherland1998.F90
Description: An implementation of Gaunt factors using the Sutherland [1998] fitting function.
Code lines: 117
Modules used: atomic_ionization_potentials

interface: gauntfactorsutherland1998
Description: Constructors for the sutherland1998 gaunt factor class.
Code lines: 4
Contained by: file atomic.radiation.gaunt_factors.Sutherland1998.F90

function: sutherland1998constructorinternal
Description: Internal constructor for the sutherland1998 gaunt factor class.
Code lines: 8
Contained by: file atomic.radiation.gaunt_factors.Sutherland1998.F90

function: sutherland1998constructorparameters
Description: Constructor for the sutherland1998 gaunt factor class which takes a parameter set as input.
Code lines: 13
Contained by: file atomic.radiation.gaunt_factors.Sutherland1998.F90
Modules used: input_parameters

subroutine: sutherland1998destructor
Description: Destructor for the sutherland1998 gaunt factor class.
Code lines: 7
Contained by: file atomic.radiation.gaunt_factors.Sutherland1998.F90

function: sutherland1998total

Description: Compute thermally averaged Gaunt factors for thermal electron distributions using the tabulations and fits of Sutherland [1998].
Code lines: 35
Contained by: file atomic.radiation.gaunt_factors.Sutherland1998.F90
Modules used: arrays_search, galacticus_error, iso_c_binding, numerical_constants_physical, numerical_constants_units

file: atomic.radiation.gaunt_factors.vanHoof2014.F90
Description: An implementation of Gaunt factors using the van Hoof et al. [2014] fitting function.
Code lines: 120
Modules used: atomic_ionization_potentials

interface: gauntfactorvanhoof2014
Description: Constructors for the vanHoof2014 gaunt factor class.
Code lines: 4
Contained by: file atomic.radiation.gaunt_factors.vanHoof2014.F90

function: vanhoof2014constructorinternal
Description: Internal constructor for the vanHoof2014 gaunt factor class.
Code lines: 8
Contained by: file atomic.radiation.gaunt_factors.vanHoof2014.F90

function: vanhoof2014constructorparameters
Description: Constructor for the vanHoof2014 gaunt factor class which takes a parameter set as input.
Code lines: 13
Contained by: file atomic.radiation.gaunt_factors.vanHoof2014.F90
Modules used: input_parameters

subroutine: vanhoof2014destructor
Description: Destructor for the vanHoof2014 gaunt factor class.
Code lines: 7
Contained by: file atomic.radiation.gaunt_factors.vanHoof2014.F90

function: vanhoof2014total
Description: Compute thermally averaged Gaunt factors for thermal electron distributions using the tabulations and fits of van Hoof et al. [2014].
Code lines: 38
Contained by: file atomic.radiation.gaunt_factors.vanHoof2014.F90
Modules used: arrays_search, galacticus_error, iso_c_binding, numerical_constants_physical, numerical_constants_units

file: atomic.rates.excitation.collisional.F90
Description: Contains a module which provides a class implenting atomic collisional excitation rates.
Code lines: 42

module: atomic_rates_excitation_collisional
Description: Provides a class implenting atomic collisional excitation rates.
Code lines: 18

Contained by: file atomic.rates.excitation.collisional.F90
Used by: subroutine galacticus_function_classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; file universe.operators.intergalactic_medium_state_evolve.F90

file: atomic.rates.excitation.collisional.ScholzWalters91.F90
Description: An implementation of atomic collisional excitation using the fitting functions of Scholz and Walters [1991].
Code lines: 98

interface: atomicexcitationratecollisionalscholzwalters1991
Description: Constructors for the scholzWalters1991 atomic collisional excitation class.
Code lines: 3
Contained by: file atomic.rates.excitation.collisional.ScholzWalters91.F90

function: scholzwalters1991constructorparameters
Description: Constructor for the scholzWalters1991 atomic collisional excitation class which takes a parameter set as input.
Code lines: 11
Contained by: file atomic.rates.excitation.collisional.ScholzWalters91.F90
Modules used: input_parameters

function: scholzwalters1991coolingrate
Description: Return collisional excitation cooling rates, in units of J/m^3/s, for ion Ion at temperature T (in Kelvin) using the fitting functions of Scholz and Walters [1991].
Code lines: 44
Contained by: file atomic.rates.excitation.collisional.ScholzWalters91.F90
Modules used: galacticus_error

file: atomic.rates.ionization.collisional.F90
Description: Contains a module which provides a class implenting atomic collisional ionization rates.
Code lines: 40

module: atomic_rates_ionization_collisional
Description: Provides a class implenting radiative recombiantion rates.
Code lines: 18
Contained by: file atomic.rates.ionization.collisional.F90
Used by: file chemical.reaction_rates.hydrogen.F90; subroutine galacticus_function_classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; file universe.operators.intergalactic_medium_state_evolve.F90

file: atomic.rates.ionization.collisional.Verner.F90
Description: An implementation of atomic collisional ionization rates based on the code originally written by Dima Verner.
Code lines: 497

interface: atomicionizationratecollisionalverner1996

Description: Constructors for the `verner1996` atomic collisional ionization class.
Code lines: 3
Contained by: file `atomic.rates.ionization.collisional.Verner.F90`

function: `verner1996constructorparameters`
Description: Constructor for the `verner1996` atomic collisional ionization class which takes a parameter set as input.
Code lines: 11
Contained by: file `atomic.rates.ionization.collisional.Verner.F90`
Modules used: `input_parameters`

function: `verner1996rate`
Description: Computes the rate coefficient of direct collisional ionization (in units of $\mathrm{cm}^3\ \mathrm{s}^{-1}$) at the specified `temperature` for all ions of atoms with $Z < 28$ by use of the fits from Voronov (1997; Version 2, March 24, 1997). Based on the code originally written by Dima Verner. The ionization state passed to this function should be that of the atom/ion prior to ionization.
Code lines: 32
Contained by: file `atomic.rates.ionization.collisional.Verner.F90`
Modules used: `numerical_constants_physical` `numerical_constants_units`

file: `atomic.rates.recombination.dielectronic.Arnaud85.F90`
Description: Implements an atomic dielectronic recombination class which uses the fits from Aldrovandi and Pequignot [1973], Shull and van Steenberg [1982] and Arnaud and Rothenflug [1985].
Code lines: 251

function: `arnaud1985constructorparameters`
Description: Constructor for the `arnaud1985` atomic ionization potentail class which builds the object from a parameter set.
Code lines: 11
Contained by: file `atomic.rates.recombination.dielectronic.Arnaud85.F90`
Modules used: `input_parameters`

function: `arnaud1985rate`
Description: Calculates rates of dielectric recombination for all ionization stages of all elements from H to Ni ($Z = 28$) by use of the fits from Aldrovandi and Pequignot [1973], Shull and van Steenberg [1982] and Arnaud and Rothenflug [1985]. Input parameters: `atomicNumber`: atomic number; `electronNumber`: number of electrons; `temperature`: temperature [K]. Output parameter: rate coefficient [$\mathrm{cm}^3\ \mathrm{s}^{-1}$].
Code lines: 22
Contained by: file `atomic.rates.recombination.dielectronic.Arnaud85.F90`
Modules used: `galacticus_error`

interface: `atomicrecombinationratedielectronicarnaud1985`
Description: Constructors for the `arnaud1985` atomic dielectronic recombination rate class.
Code lines: 3
Contained by: file `atomic.rates.recombination.dielectronic.Arnaud85.F90`

file: `atomic.rates.recombination.dielectronic.F90`
Description: Contains a module that implements a dielectronic recombination rate class.
Code lines: 42

module: `atomic_rates_recombination_dielectronic`

Description: Implements a dielectronic recombination rates class.
Code lines: 18
Contained by: file atomic.rates.recombination.dielectronic.F90
Used by: subroutine galacticus_function_-classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; file universe.operators.intergalactic_-medium_state_evolve.F90

file: `atomic.rates.recombination.radiative.F90`

Description: Contains a module which provides a class implenting atomic radiative recombination rates.
Code lines: 49

module: `atomic_rates_recombination_radiative`

Description: Provides a class implenting radiative recombiantion rates.
Code lines: 27
Contained by: file atomic.rates.recombination.radiative.F90
Used by: file chemical.reaction_-rates.hydrogen.F90; subroutine galacticus_function_-classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; file universe.operators.intergalactic_-medium_state_evolve.F90

file: `atomic.rates.recombination.radiative.Verner.F90`

Description: An implementation of atomic radiative recombination rates based on the code originally written by Dima Verner.
Code lines: 655

interface: `atomicrecombinationrateradiativeverner1996`

Description: Constructors for the `verner1996` atomic radiative recombination class.
Code lines: 3
Contained by: file atomic.rates.recombination.radiative.Verner.F90

function: `verner1996constructorparameters`

Description: Constructor for the `verner1996` atomic radiative recombination class which takes a parameter set as input.
Code lines: 10
Contained by: file atomic.rates.recombination.radiative.Verner.F90
Modules used: input_parameters

function: `verner1996rate`

Description: Computes the rate coefficient of radiative recombination (in units of $\mathrm{cm}^3\ \mathrm{s}^{-1}$) at the specified `temperature` for all ions of all elements from H through Zn (selected by the `atomicNumber` and the `ionizationState` *of the recombined ion*) use of the following fits:

- H-like, He-like, Li-like, Na-like: [Verner and Ferland, 1996];
- Other ions of C, N, O, Ne: [Pequignot et al., 1991], refitted by Verner & Ferland formula to ensure correct asymptotes;
- Fe XVII-XXIII: [Arnaud and Raymond, 1992];
- Fe I-XV: refitted by Verner & Ferland formula to ensure correct asymptotes;
- Other ions of Mg, Si, S, Ar, Ca, Fe, Ni: [Shull and van Steenberg, 1982];
- Other ions of Na, Al: [Landini and Fossi, 1990];
- Other ions of F, P, Cl, K, Ti, Cr, Mn, Co (excluding Ti I-II, Cr I-IV, Mn I-V, Co I): [Landini and Fossi, 1991];
- All other species: interpolations of the power-law fits.

Based on the code originally written by Dima Verner. The ionization state passed to this function should be that of the atom/ion post recombination.

Code lines: 104
Contained by: file `atomic.rates.recombination.radiative.Verner.F90`
Modules used: `galacticus_error`

file: `benchmarks.stellar_luminosities.F90`

Description: Contains a program to benchmark stellar population luminosity calculations.
Code lines: 124

program: `benchmark_stellar_populations_luminosities`
Description: Benchmarking of stellar population luminosity calculations.
Code lines: 100
Contained by: file benchmarks.stellar_luminosities.F90
Modules used: abundances_structure, cosmology_functions, cosmology_parameters, galacticus_display, galacticus_paths, input_parameters, instruments_filters, iso_varying_string, kind_numbers, stellar_astrophysics, stellar_astrophysics_tracks, stellar_astrophysics_winds, stellar_feedback, stellar_population_luminosities, stellar_population_spectra, stellar_population_spectra_postprocess, stellar_populations, stellar_populations_initial_mass_functions, supernovae_population_iii, supernovae_type_ia

file: `bivar.F90`
Code lines: 3004

module: `bivar`
Code lines: 3003
Contained by: file bivar.F90
Used by: function interpolate_2d_irregular_array

subroutine: `idbvip`
Code lines: 287
Contained by: module bivar

subroutine: `idcldp`
Code lines: 225
Contained by: module bivar

subroutine: `idlctn`
Code lines: 355
Contained by: module bivar
Modules used: omp_lib

subroutine: `idpdrv`
Code lines: 245
Contained by: module bivar

subroutine: `idptip`
Code lines: 470
Contained by: module bivar

subroutine: `idtang`
Code lines: 589
Contained by: module `bivar`

function: `idxchg`
Code lines: 132
Contained by: module `bivar`

file: `black_holes.binaries.initial_separation.F90`
Description: Contains a module which implements a class for calculations of black hole binary initial separations.
Code lines: 41

module: `black_hole_binary_initial_separation`
Description: Implements a class for black hole binary initial separations.
Code lines: 19
Contained by: file `black_holes.binaries.initial_separation.F90`
Modules used: `galacticus_nodes`
Used by: subroutine `galacticus_function_classes_destroy`; subroutine `galacticus_state_retrieve`; subroutine `galacticus_state_store`; module `node_component_black_hole_standard`

file: `black_holes.binaries.initial_separation.Volonteri_2003.F90`
Description: Implements a class for black hole binary initial separation based on the model of Volonteri et al. [2003].
Code lines: 97
Modules used: `dark_matter_halo_scales`

interface: `blackholebinaryinitialseparationvolonteri2003`
Description: Constructors for the `volonteri2003` black hole binary initial radius class.
Code lines: 4
Contained by: file `black_holes.binaries.initial_separation.Volonteri_2003.F90`

function: `volonteri2003constructorinternal`
Description: Constructor for the `volonteri2003` black hole bianry initial radius class which takes a parameter list as input.
Code lines: 10
Contained by: file `black_holes.binaries.initial_separation.Volonteri_2003.F90`
Modules used: `input_parameters`

function: `volonteri2003constructorparameters`
Description: Constructor for the `volonteri2003` black hole bianry initial radius class which takes a parameter list as input.
Code lines: 14
Contained by: file `black_holes.binaries.initial_separation.Volonteri_2003.F90`
Modules used: `input_parameters`

subroutine: `volonteri2003destructor`

Description: Destructor for the `volonteri2003` black hole binary initial separation class.
Code lines: 7
Contained by: file `black_holes.binaries.initial_separation.Volonteri_2003.F90`

function: `volonteri2003separationinitial`
Description: Returns an initial separation for binary black holes using the method of Volonteri et al. [2003], with the assumption that the local velocity dispersion is approximately the dark matter halo virial velocity.
Code lines: 14
Contained by: file `black_holes.binaries.initial_separation.Volonteri_2003.F90`
Modules used: `galacticus_nodes` `numerical_constants_physical`

file: `black_holes.binaries.initial_separation.spheroid_size_fraction.F90`
Description: Implements a class for black hole binary initial separation in which the radius is a fixed fraction of the scale radius of the larger of the host and satellite spheroids.
Code lines: 89

interface: `blackholebinaryinitialseparationspheroidradiusfraction`
Description: Constructors for the `spheroidRadiusFraction` black hole binary initial radius class.
Code lines: 4
Contained by: file `black_holes.binaries.initial_separation.spheroid_size_fraction.F90`

function: `spheroidradiusfractionconstructorinternal`
Description: Constructor for the `spheroidRadiusFraction` black hole bianry recoild class which takes a parameter list as input.
Code lines: 10
Contained by: file `black_holes.binaries.initial_separation.spheroid_size_fraction.F90`
Modules used: `input_parameters`

function: `spheroidradiusfractionconstructorparameters`
Description: Constructor for the `spheroidRadiusFraction` black hole bianry recoild class which takes a parameter list as input.
Code lines: 20
Contained by: file `black_holes.binaries.initial_separation.spheroid_size_fraction.F90`
Modules used: `input_parameters`

function: `spheroidradiusfractionseparationinitial`
Description: Returns an initial separation for a binary black holes that is a fixed fraction of the scale radius of the larger of the host and satellite spheroids.
Code lines: 13
Contained by: file `black_holes.binaries.initial_separation.spheroid_size_fraction.F90`
Modules used: `galacticus_nodes`

file: `black_holes.binaries.initial_separation.tidal_radius.F90`
Description: Implements a class for black hole binary initial separation based on tidal disruption of the satellite galaxy.
Code lines: 107

interface: `blackholebinaryinitialseparationtidalradius`

Description: Constructors for the `tidalRadius` black hole binary initial radius class.
Code lines: 3
Contained by: file `black_holes.binaries.initial_separation.tidal_radius.F90`

function: `tidalradiusconstructorparameters`
Description: Constructor for the `tidalRadius` black hole bianry recoild class which takes a parameter list as input.
Code lines: 11
Contained by: file `black_holes.binaries.initial_separation.tidal_radius.F90`
Modules used: `input_parameters`

function: `tidalradiusroot`
Description: Root function used in solving for the radius of tidal disruption of a satellite galaxy.
Code lines: 10
Contained by: file `black_holes.binaries.initial_separation.tidal_radius.F90`
Modules used: `galactic_structure_enclosed_masses` `galactic_structure_options`

function: `tidalradiusseparationinitial`
Description: Returns an initial separation for a binary black holes through tidal disruption.
Code lines: 36
Contained by: file `black_holes.binaries.initial_separation.tidal_radius.F90`
Modules used: `galactic_structure_enclosed_masses` `galactic_structure_options` `galacticus_nodes` `root_finder`

file: `black_holes.binaries.recoil_velocity.Campanelli2007.F90`
Description: Implements a black hole binary recoil velocity class which follows the formulae in Campanelli et al. [2007], derived from post-Newtonian evaluations.
Code lines: 95

interface: `blackholebinaryrecoilcampanelli2007`
Description: Constructors for the `campanelli2007` black hole binary recoil class.
Code lines: 3
Contained by: file `black_holes.binaries.recoil_velocity.Campanelli2007.F90`

function: `campanelli2007constructorparameters`
Description: Constructor for the `campanelli2007` black hole bianry recoild class which takes a parameter list as input.
Code lines: 11
Contained by: file `black_holes.binaries.recoil_velocity.Campanelli2007.F90`
Modules used: `input_parameters`

function: `campanelli2007velocity`

Description:	Returns the recoil velocity of a black hole binary, accounting for the parallel and perpendicular velocities, plus that of the binary's center of mass. Constants used are retrieved from the articles by: Koppitz et al. [2007] for $H = (7.3 \pm 0.3)10^3$ km/s, González et al. [2007b] for $A = 1.2 \times 10^4$ km/s $B = -0.93$, González et al. [2007a] for $K \cos(\delta\theta) = (6, -5.3)10^4$ km/s and $K = (6.0 \pm 0.1)10^4$ km/s.
Code lines:	38
Contained by:	file `black_holes.binaries.recoil_velocity.Campanelli2007.F90`
Modules used:	`numerical_constants_math`

file: `black_holes.binaries.recoil_velocity.F90`

Description: Contains a module which implements a class for black hole binary recoil velocities.
Code lines: 43

module: `black_hole_binary_recoil_velocities`
Description: Implements a class for black hole binary recoil velocities.
Code lines: 19
Contained by: file black_holes.binaries.recoil_velocity.F90
Modules used: galacticus_nodes
Used by: subroutine galacticus_function_classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; module node_component_black_hole_noncentral; module node_component_black_hole_standard

file: `black_holes.binaries.recoil_velocity.zero.F90`
Description: Implements a black hole binary recoil velocity class in which the recoil velocity is zero.
Code lines: 62

interface: `blackholebinaryrecoilzero`
Description: Constructors for the `zero` black hole binary recoil class.
Code lines: 3
Contained by: file black_holes.binaries.recoil_velocity.zero.F90

function: `zeroconstructorparameters`
Description: Constructor for the `zero` black hole binary recoil class which takes a parameter list as input.
Code lines: 10
Contained by: file black_holes.binaries.recoil_velocity.zero.F90
Modules used: input_parameters

function: `zerovelocity`
Description: Compute the recoil velocity for a pair of merging black holes.
Code lines: 9
Contained by: file black_holes.binaries.recoil_velocity.zero.F90

file: `black_holes.binaries.separation_growth_rate.F90`
Description: Contains a module which implements a class for calculations of black hole binary separation growth rate.
Code lines: 43

module: `black_hole_binary_separations`
Description: Implements a class for calculations of black hole binary separation growth rate.
Code lines: 19
Contained by: file black_holes.binaries.separation_growth_rate.F90
Modules used: galacticus_nodes
Used by: subroutine galacticus_function_classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; module node_component_black_hole_noncentral

module node_component_black_hole_standard

subroutine node_component_black_hole_standard_rate_compute

file: black_holes.binaries.separation_growth_rate.standard.F90
Description: Implements a black hole binary separation growth class which follows a modified version of Volonteri et al. [2003], including terms for dynamical friction, hardening due to scattering of stars and emission of gravitational waves.
Code lines: 218
Modules used: dark_matter_halo_scales

interface: blackholebinaryseparationgrowthratestandard
Description: Constructors for the standard black hole binary recoil class.
Code lines: 4
Contained by: file black_holes.binaries.separation_growth_rate.standard.F90

function: standardconstructorinternal
Description: Internal constructor for the standard black hole binary separation growth class.
Code lines: 9
Contained by: file black_holes.binaries.separation_growth_rate.standard.F90

function: standardconstructorparameters
Description: Constructor for the standard black hole binary separation growth rate class which takes a parameter set as input.
Code lines: 33
Contained by: file black_holes.binaries.separation_growth_rate.standard.F90
Modules used: input_parameters

subroutine: standarddestructor
Description: Destructor for the standard black hole binary separation growth class.
Code lines: 7
Contained by: file black_holes.binaries.separation_growth_rate.standard.F90

function: standardgrowthrate
Description: Returns an initial separation growth rate for a binary black holes that follows a modified version of Volonteri et al. [2003].
Code lines: 114
Contained by: file black_holes.binaries.separation_growth_rate.standard.F90
Modules used: galactic_structure_densities, galactic_structure_options, galactic_structure_rotation_curve_gradients, galactic_structure_rotation_curves, galactic_structure_velocity_dispersions, galacticus_display, galacticus_error, galacticus_nodes, numerical_constants_astronomical, numerical_constants_physical

file: black_holes.binaries.separation_growth_rate.zero.F90
Description: Implements a black hole binary separation growth class in which the separation does not grow.
Code lines: 63

interface: `blackholebinaryseparationgrowthratezero`
Description: Constructors for the `zero` black hole binary separation growth rate class.
Code lines: 3
Contained by: file `black_holes.binaries.separation_growth_rate.zero.F90`

function: `zeroconstructorparameters`
Description: Constructor for the `zero` black hole binary separation growth rate class which takes a parameter set as input.
Code lines: 11
Contained by: file `black_holes.binaries.separation_growth_rate.zero.F90`
Modules used: `input_parameters`

function: `zerogrowthrate`
Description: Returns a separation growth rate for a binary black hole that is always zero.
Code lines: 9
Contained by: file `black_holes.binaries.separation_growth_rate.zero.F90`

file: `black_holes.binary_mergers.F90`
Description: Contains a module which implements a class for calculations of black hole binary mergers.
Code lines: 41

module: `black_hole_binary_mergers`
Description: Implements a class for calculations of black hole binary mergers.
Code lines: 19
Contained by: file `black_holes.binary_mergers.F90`
Used by:
- subroutine `galacticus_function_classes_destroy`
- subroutine `galacticus_state_retrieve`
- subroutine `galacticus_state_store`
- module `node_component_black_hole_noncentral`
- module `node_component_black_hole_simple`
- module `node_component_black_hole_standard`

file: `black_holes.binary_mergers.Rezzolla2008.F90`
Description: Implements a black hole binary merger in which the black hole mass and spin resulting from binary mergers utilizing the approximations of Rezzolla et al. [2008].
Code lines: 96

interface: `blackholebinarymergerrezzolla2008`
Description: Constructors for the `rezzolla2008` black hole binary merger class.
Code lines: 3
Contained by: file `black_holes.binary_mergers.Rezzolla2008.F90`

function: `rezzolla2008constructorparameters`
Description: Constructor for the `rezzolla2008` black hole bianry merger class which takes a parameter list as input.
Code lines: 11
Contained by: file `black_holes.binary_mergers.Rezzolla2008.F90`
Modules used: `input_parameters`

subroutine: `rezzolla2008merge`
Description: Computes the mass and spin of a black hole resulting from a binary merger utilizing the approximations of Rezzolla et al. [2008].
Code lines: 43
Contained by: file `black_holes.binary_mergers.Rezzolla2008.F90`

file: `black_holes.fundamentals.F90`
Description: Contains a module which implements fundamental properties of black holes.
Code lines: 575

module: `black_hole_fundamentals`
Description: Implements fundamental properties of black holes.
Code lines: 553
Contained by: file `black_holes.fundamentals.F90`
Used by:

function `adafconstructorinternal`	function `adafenthalpyangularmomentumproduct`
function `adaffieldenhancement`	function `adaffluidangularvelocity`
function `adafgammaazimuthal`	function `adafheight`
function `adafjetpowerblackhole`	function `adafjetpowerdisk`
function `adafpowerjet`	function `adaftemperature`
function `adafvelocity`	function `eddingtonlimitedpowerjet`
function `shakurasunyaevefficiencyradiative`	function `shakurasunyaevpowerjet`
function `shakurasunyaevratespinup`	function `switchedefficiencyradiativescalingadaf`
function `switchedfractionadaf`	function `node_component_black_hole_simple_potential`
function `node_component_black_hole_simple_rotation_curve`	function `node_component_black_hole_simple_rotation_curve_gradient`
subroutine `node_component_black_hole_standard_mass_accretion_rate`	function `node_component_black_hole_standard_potential`
function `node_component_black_hole_standard_rotation_curve`	function `node_component_black_hole_standard_rotation_curve_gradient`
program `test_black_hole_fundamentals`	

function: `a1`
Description: Return the function $A_1(j)$ that appears in the Kerr metric with spin `blackHoleSpin`.
Code lines: 7
Contained by: module `black_hole_fundamentals`

function: `a2`
Description: Return the function $A_2(j)$ that appears in the Kerr metric with spin `blackHoleSpin`.
Code lines: 7
Contained by: module `black_hole_fundamentals`

function: `black_hole_eddington_accretion_rate`
Description: Return the Eddington accretion rate (in $M_\odot$ Gyr^{-1}) for the black hole in `thisBlackHole`.
Code lines: 10

Contained by: module black_hole_fundamentals
Modules used: galacticus_nodes numerical_constants_astronomical numerical_constants_physical

interface: black_hole_frame_dragging_frequency
Code lines: 3
Contained by: module black_hole_fundamentals

function: black_hole_frame_dragging_frequency_node
Description: Returns the frame-dragging angular velocity in the Kerr metric.
Code lines: 34
Contained by: module black_hole_fundamentals
Modules used: galacticus_error galacticus_nodes

function: black_hole_frame_dragging_frequency_spin
Description: Returns the frame-dragging angular velocity in the Kerr metric.
Code lines: 7
Contained by: module black_hole_fundamentals

function: black_hole_gravitational_radius
Description: Computes the gravitational radius (in Mpc) for the thisBlackHole.
Code lines: 9
Contained by: module black_hole_fundamentals
Modules used: galacticus_nodes numerical_constants_physical

interface: black_hole_horizon_radius
Code lines: 3
Contained by: module black_hole_fundamentals

function: black_hole_horizon_radius_node
Description: Return the radius of the horizon for a Kerr metric with dimensionless angular momentum j. The radius is in units of the gravitational radius.
Code lines: 32
Contained by: module black_hole_fundamentals
Modules used: galacticus_error galacticus_nodes

function: black_hole_horizon_radius_spin
Description: Return the radius of the horizon for a Kerr metric with dimensionless angular momentum j. The radius is in units of the gravitational radius.
Code lines: 8
Contained by: module black_hole_fundamentals

interface: black_hole_isco_radius
Code lines: 3
Contained by: module black_hole_fundamentals

function: black_hole_isco_radius_node
Description: Returns the radius (in physical or gravitational units and for a prograde or retorgrade orbit) of the innermost stable circular orbit for the black hole in thisBlackHole.
Code lines: 33

Contained by: module black_hole_fundamentals
Modules used: galacticus_nodes

function: black_hole_isco_radius_spin
Description: Returns the radius (in gravitational units and for a prograde or retorgrade orbit) of the innermost stable circular orbit for a black hole with spin blackHoleSpin.
Code lines: 42
Contained by: module black_hole_fundamentals
Modules used: galacticus_error

function: black_hole_isco_specific_angular_momentum
Description: Returns the specific angular momentum (in physical or gravitational units and for a prograde or retorgrade orbit) of the innermost stable circular orbit for the black hole in thisBlackHole.
Code lines: 43
Contained by: module black_hole_fundamentals
Modules used: galacticus_nodes numerical_constants_physical

interface: black_hole_isco_specific_energy
Code lines: 3
Contained by: module black_hole_fundamentals

function: black_hole_isco_specific_energy_node
Description: Returns the specific energy (in physical or gravitational units and for a prograde or retorgrade orbit) of the innermost stable circular orbit for the black hole in thisNode.
Code lines: 27
Contained by: module black_hole_fundamentals
Modules used: galacticus_nodes numerical_constants_physical

function: black_hole_isco_specific_energy_spin
Description: Returns the specific energy (in physical or gravitational units and for a prograde or retorgrade orbit) of the innermost stable circular orbit for a black hole of given blackHoleSpin.
Code lines: 23
Contained by: module black_hole_fundamentals

interface: black_hole_metric_a_factor
Code lines: 3
Contained by: module black_hole_fundamentals

function: black_hole_metric_a_factor_node
Description: Returns the $\mathcal{A}$ factor appearing in the Kerr metric for thisBlackHole.
Code lines: 34
Contained by: module black_hole_fundamentals
Modules used: galacticus_error galacticus_nodes

function: black_hole_metric_a_factor_spin
Description: Returns the $\mathcal{A}$ factor appearing in the Kerr metric for spin blackHoleSpin.
Code lines: 7
Contained by: module black_hole_fundamentals

interface: `black_hole_metric_d_factor`
Code lines: 3
Contained by: module black_hole_fundamentals

function: `black_hole_metric_d_factor_node`
Description: Returns the $\mathcal{D}$ factor appearing in the Kerr metric for `thisBlackHole`.
Code lines: 34
Contained by: module black_hole_fundamentals
Modules used: galacticus_error galacticus_nodes

function: `black_hole_metric_d_factor_spin`
Description: Returns the $\mathcal{D}$ factor appearing in the Kerr metric for spin `blackHoleSpin`.
Code lines: 7
Contained by: module black_hole_fundamentals

interface: `black_hole_rotational_energy_spin_down`
Code lines: 3
Contained by: module black_hole_fundamentals

function: `black_hole_rotational_energy_spin_down_node`
Description: Wrapper function for Black_Hole_Rotational_Energy_Spin_Down_Node which takes a tree node as input.
Code lines: 14
Contained by: module black_hole_fundamentals
Modules used: galacticus_nodes

function: `black_hole_rotational_energy_spin_down_spin`
Description: Computes the spin down rate of a black hole due to extraction of rotational energy. Specifically, it returns the factor S in the relation:

$$s = -S \frac{P_{\rm rotation}}{\dot{M}_{\bullet,0} {\rm c}^2}, \tag{18.1}$$

where $P_{\rm rotation}$ is the power of rotational energy extraction and

$$S = [(1+\sqrt{1-j^2})^2 + j^2] \frac{\sqrt{1-j^2}}{j}, \tag{18.2}$$

for black hole spin j.
Code lines: 30
Contained by: module black_hole_fundamentals

interface: `black_hole_static_radius`
Code lines: 3
Contained by: module black_hole_fundamentals

function: `black_hole_static_radius_node`
Description: Return the radius of the static limit for a Kerr metric for the black hole in `thisBlackHole` and angle `theta`.
Code lines: 35

Contained by: module black_hole_fundamentals
Modules used: galacticus_error galacticus_nodes

function: black_hole_static_radius_spin
Description: Return the radius of the static limit for a Kerr metric for a black hole of given blackHoleSpin and angle theta.
Code lines: 18
Contained by: module black_hole_fundamentals
Modules used: numerical_constants_math

file: chemical.reaction_rates.F90
Description: Contains a module that implements a class providing calculations of chemical reaction rates.
Code lines: 46

module: chemical_reaction_rates
Description: Provides a class implementing chemical reaction rates.
Code lines: 24
Contained by: file chemical.reaction_rates.F90
Modules used: chemical_abundances_structure galacticus_nodes radiation_fields
Used by: subroutine galacticus_function_classes_destroy subroutine galacticus_state_retrieve
subroutine galacticus_state_store module node_component_hot_halo_standard

file: chemical.reaction_rates.hydrogen.F90
Description: An implementation of calculations of chemical reaction rates for hydrogen using the fits from Abel et al. [1997] and Tegmark et al. [1997].
Code lines: 1652
Modules used: atomic_cross_sections_ionization_photo atomic_rates_ionization_collisional
atomic_rates_recombination_radiative

interface: chemicalreactionratehydrogennetwork
Description: Constructors for the hydrogenNetwork chemical reaction rates class.
Code lines: 4
Contained by: file chemical.reaction_rates.hydrogen.F90

function: hydrogennetworkconstructorinternal
Description: Constructor for the hydrogenNetwork chemical reaction rates class which takes a parameter set as input.
Code lines: 26
Contained by: file chemical.reaction_rates.hydrogen.F90
Modules used: galacticus_error

function: hydrogennetworkconstructorparameters
Description: Constructor for the hydrogenNetwork chemical reaction rates class which takes a parameter set as input.
Code lines: 31
Contained by: file chemical.reaction_rates.hydrogen.F90

Modules used: input_parameters

function: hydrogennetworkcrosssection_h2_gamma_to_2h

Description: Compute the cross-section (in units of cm^2) for the reaction $\mathrm{H}_2 + \gamma \rightarrow 2\mathrm{H}$ as given by Abel et al. [1997].

Code lines: 40

Contained by: file chemical_reaction_rates.hydrogen.F90

Modules used: numerical_constants_physical numerical_constants_units

function: hydrogennetworkcrosssection_h2_gamma_to_h2plus_electron

Description: Compute the cross-section (in units of cm^2) for the reaction $\mathrm{H}_2 + \gamma \rightarrow \mathrm{H}_2^+ + \mathrm{e}^-$ as given by[a] Abel et al. [1997].

[a] Abel et al. [1997] cite "O'Neil & Reinhardt (1978)" as the source for this fit, but it is not listed in their bibliography, and I have not been able to locate by any other means.

Code lines: 24

Contained by: file chemical_reaction_rates.hydrogen.F90

Modules used: numerical_constants_physical numerical_constants_units

function: hydrogennetworkcrosssection_h2plus_gamma_to_2hplus_electron

Description: Compute the cross-section (in units of cm^2) for the reaction $\mathrm{H}_2^+ + \gamma \rightarrow 2\mathrm{H}^+ + \mathrm{e}^-$ as given by Shapiro and Kang [1987].

Code lines: 18

Contained by: file chemical_reaction_rates.hydrogen.F90

Modules used: numerical_constants_physical numerical_constants_units

function: hydrogennetworkcrosssection_h2plus_gamma_to_h_hplus

Description: Compute the cross-section (in units of cm^2) for the reaction $\mathrm{H}_2^+ + \gamma \rightarrow \mathrm{H} + \mathrm{H}^+$ as given by Shapiro and Kang [1987].

Code lines: 20

Contained by: file chemical_reaction_rates.hydrogen.F90

Modules used: numerical_constants_physical numerical_constants_units

function: hydrogennetworkcrosssection_h_gamma_to_hplus_electron

Description: Compute the cross-section (in units of cm^2) for the reaction $\mathrm{H}_2 + \gamma \rightarrow 2\mathrm{H}$ as given by Abel et al. [1997].

Code lines: 9

Contained by: file chemical_reaction_rates.hydrogen.F90

function: hydrogennetworkcrosssection_hminus_gamma_to_h_electron

Description: Compute the cross-section (in units of cm^2) for the reaction $\mathrm{H}^- + \gamma \rightarrow \mathrm{H} + \mathrm{e}^-$ using the fitting function given by Shapiro and Kang [1987], renormalized[a] to match the results of Nascimento and Goddard [1977].

[a] It seems unclear what units were used in Shapiro and Kang [1987], hence the recalibration.

Code lines: 21

Contained by: file chemical_reaction_rates.hydrogen.F90

Modules used: numerical_constants_physical numerical_constants_units

subroutine: hydrogennetworkdestructor

Description: Destructor for the `hydrogenNetwork` chemical reaction rates class.
Code lines: 9
Contained by: file chemical.reaction_rates.hydrogen.F90

function: `hydrogennetworkh_electron_to_hminus_photon_ratecoefficient`
Description: Computes the rate coefficient (in units of $\mathrm{cm}^3\ \mathrm{s}^{-1}$) for the reaction $\mathrm{H} + \mathrm{e}^- \rightarrow \mathrm{H}^- + \gamma$.
Code lines: 26
Contained by: file chemical.reaction_rates.hydrogen.F90

function: `hydrogennetworkh_hminus_to_h2_electron_ratecoefficient`
Description: Computes the rate coefficient (in units of $\mathrm{c}^3\ \mathrm{s}^{-1}$) for the reaction $\mathrm{H} + \mathrm{H}^- \rightarrow \mathrm{H}_2 + \mathrm{e}^-$.
Code lines: 19
Contained by: file chemical.reaction_rates.hydrogen.F90
Modules used: numerical_constants_physical numerical_constants_units

function: `hydrogennetworkhminus_electron_to_h_2electron_ratecoefficient`
Description: Computes the rate coefficient (in units of $\mathrm{cm}^3\ \mathrm{s}^{-1}$) for the reaction $\mathrm{H}_2^+ + \mathrm{H} \rightarrow \mathrm{H}_2 + \mathrm{H}^+$.
Code lines: 14
Contained by: file chemical.reaction_rates.hydrogen.F90
Modules used: numerical_constants_physical numerical_constants_units

function: `hydrogennetworkhminus_hplus_to_2h_ratecoefficient`
Description: Compute the rate coefficient (in units of $\mathrm{c}^3\ \mathrm{s}^{-1}$) for the reaction $\mathrm{H}_2^+ + \mathrm{H} \rightarrow \mathrm{H}_2 + \mathrm{H}^+$.
Code lines: 7
Contained by: file chemical.reaction_rates.hydrogen.F90

subroutine: `hydrogennetworkrateh2_electron_to_2h_electron`
Description: Computes the rate (in units of $\mathrm{c}^{-3}\ \mathrm{s}^{-1}$) for the reaction $\mathrm{H}_2 + \mathrm{e}^- \rightarrow 2\mathrm{H} + \mathrm{e}^-$.
Code lines: 44
Contained by: file chemical.reaction_rates.hydrogen.F90
Modules used: radiation_fields

subroutine: `hydrogennetworkrateh2_gamma_to_2h`
Description: Computes the rate (in units of $\mathrm{cm}^{-3}\ \mathrm{s}^{-1}$) for the reaction $\mathrm{H}_2^+ + \gamma \rightarrow 2\mathrm{H}^+ + \mathrm{e}^-$.
Code lines: 49
Contained by: file chemical.reaction_rates.hydrogen.F90
Modules used: numerical_constants_physical numerical_constants_units radiation_fields

subroutine: `hydrogennetworkrateh2_gamma_to_h2plus_electron`
Description: Computes the rate (in units of $\mathrm{cm}^{-3}\ \mathrm{s}^{-1}$) for the reaction $\mathrm{H}_2 + \gamma \rightarrow \mathrm{H}_2^+ + \mathrm{e}^-$.
Code lines: 49
Contained by: file chemical.reaction_rates.hydrogen.F90
Modules used: numerical_constants_physical numerical_constants_units radiation_fields

subroutine: `hydrogennetworkrateh2_gamma_to_h2star_to_2h`
Description: Computes the rate (in units of $\mathrm{cm}^{-3}\ \mathrm{s}^{-1}$) for the reaction $\mathrm{H}_2 + \gamma \rightarrow H_2^* \rightarrow 2\mathrm{H}$.
Code lines: 47

Contained by: file chemical.reaction_rates.hydrogen.F90
Modules used: numerical_constants_physical numerical_constants_units radiation_fields

subroutine: hydrogennetworkrateh2_h_to_3h
Description: Computes the rate (in units of c^{-3} s^{-1}) for the reaction $H_2 + H \rightarrow 3H$.
Code lines: 51
Contained by: file chemical.reaction_rates.hydrogen.F90
Modules used: numerical_constants_physical numerical_constants_units radiation_fields

subroutine: hydrogennetworkrateh2_hplus_to_h2plus_h
Description: Computes the rate (in units of c^{-3} s^{-1}) for the reaction $H_2 + H^+ \rightarrow H_2^+ + H$.
Code lines: 52
Contained by: file chemical.reaction_rates.hydrogen.F90
Modules used: numerical_constants_physical numerical_constants_units radiation_fields

subroutine: hydrogennetworkrateh2plus_electron_to_2h
Description: Computes the rate (in units of c^{-3} s^{-1}) for the reaction $H_2^+ + e^- \rightarrow 2H$.
Code lines: 46
Contained by: file chemical.reaction_rates.hydrogen.F90
Modules used: radiation_fields

subroutine: hydrogennetworkrateh2plus_gamma_to_2hplus_electron
Description: Computes the rate (in units of cm^{-3} s^{-1}) for the reaction $H_2^+ + \gamma \rightarrow 2H^+ + e^-$.
Code lines: 51
Contained by: file chemical.reaction_rates.hydrogen.F90
Modules used: numerical_constants_physical numerical_constants_units radiation_fields

subroutine: hydrogennetworkrateh2plus_gamma_to_h_hplus
Description: Computes the rate (in units of c^{-3} s^{-1}) for the reaction $H_2^+ + \gamma \rightarrow H + H^+$.
Code lines: 55
Contained by: file chemical.reaction_rates.hydrogen.F90
Modules used: numerical_constants_physical numerical_constants_units radiation_fields

subroutine: hydrogennetworkrateh2plus_h_to_h2_hplus
Description: Computes the rate (in units of c^{-3} s^{-1}) for the reaction $H_2^+ + H \rightarrow H_2 + H^+$.
Code lines: 44
Contained by: file chemical.reaction_rates.hydrogen.F90
Modules used: radiation_fields

subroutine: hydrogennetworkrateh2plus_hminus_to_h2_h
Description: Computes the rate (in units of c^{-3} s^{-1}) for the reaction $H_2^+ + H^- \rightarrow H_2 + H$.
Code lines: 45
Contained by: file chemical.reaction_rates.hydrogen.F90

Modules used: radiation_fields

subroutine: hydrogennetworkrateh_electron_to_hminus_photon
Description: Computes the rate (in units of cm^{-3} s^{-1}) for the reaction $H + e^- \rightarrow H^- + \gamma$.
Code lines: 43
Contained by: file chemical.reaction_rates.hydrogen.F90
Modules used: radiation_fields

subroutine: hydrogennetworkrateh_electron_to_hplus_2electron
Description: Computes the rate (in units of c^{-3} s^{-1}) for the reaction $H + e^- \rightarrow H^+ + 2e^-$.
Code lines: 41
Contained by: file chemical.reaction_rates.hydrogen.F90
Modules used: radiation_fields

subroutine: hydrogennetworkrateh_gamma_to_hplus_electron
Description: Computes the rate (in units of cm^{-3} s^{-1}) for the reaction $H + \gamma \rightarrow H^+ + e^-$.
Code lines: 50
Contained by: file chemical.reaction_rates.hydrogen.F90
Modules used: numerical_constants_physical numerical_constants_units radiation_fields

subroutine: hydrogennetworkrateh_hminus_to_h2_electron
Description: Computes the rate (in units of cm^{-3} s^{-1}) for the reaction $H + H^- \rightarrow H_2 + e^-$.
Code lines: 43
Contained by: file chemical.reaction_rates.hydrogen.F90
Modules used: radiation_fields

subroutine: hydrogennetworkrateh_hplus_to_h2plus_photon
Description: Computes the rate (in units of c^{-3} s^{-1}) for the reaction $H + H^+ \rightarrow H_2^+ + \gamma$.
Code lines: 51
Contained by: file chemical.reaction_rates.hydrogen.F90
Modules used: numerical_constants_physical numerical_constants_units radiation_fields

subroutine: hydrogennetworkratehminus_electron_to_h_2electron
Description: Computes the rate (in units of cm^{-3} s^{-1}) for the reaction $H_2^+ + H \rightarrow H_2 + H^+$.
Code lines: 43
Contained by: file chemical.reaction_rates.hydrogen.F90
Modules used: radiation_fields

subroutine: hydrogennetworkratehminus_gamma_to_h_electron
Description: Computes the rate (in units of c^{-3} s^{-1}) for the reaction $H^- + \gamma \rightarrow H + e^-$.
Code lines: 54
Contained by: file chemical.reaction_rates.hydrogen.F90
Modules used: numerical_constants_physical numerical_constants_units radiation_fields

subroutine: hydrogennetworkratehminus_h_to_2h_electron

Description: Computes the rate (in units of c^{-3} s^{-1}) for the reaction $H^- + H \rightarrow 2H + e^-$.
Code lines: 52
Contained by: file chemical.reaction_rates.hydrogen.F90
Modules used: numerical_constants_physical, numerical_constants_units, radiation_fields

subroutine: hydrogennetworkratehminus_hplus_to_2h
Description: Computes the rate (in units of c^{-3} s^{-1}) for the reaction $H_2^+ + H \rightarrow H_2 + H^+$.
Code lines: 43
Contained by: file chemical.reaction_rates.hydrogen.F90
Modules used: radiation_fields

subroutine: hydrogennetworkratehminus_hplus_to_h2plus_electron
Description: Computes the rate (in units of cm^{-3} s^{-1}) for the reaction $H^- + H^+ \rightarrow H_2^+ + e^-$.
Code lines: 53
Contained by: file chemical.reaction_rates.hydrogen.F90
Modules used: numerical_constants_physical, numerical_constants_units, radiation_fields

subroutine: hydrogennetworkratehplus_electron_to_h_photon
Description: Computes the rate (in units of c^{-3} s^{-1}) for the reaction $H^+ + e^- \rightarrow H + \gamma$.
Code lines: 41
Contained by: file chemical.reaction_rates.hydrogen.F90
Modules used: radiation_fields

subroutine: hydrogennetworkrates
Description: Compute rates of change of chemical abundances due to reactions involving chemical hydrogen species.
Code lines: 58
Contained by: file chemical.reaction_rates.hydrogen.F90
Modules used: galacticus_error, radiation_fields

file: chemical.reaction_rates.utilities.F90
Description: Contains a module that implements various useful utility functions for calculations of chemical abundances and rates.
Code lines: 41

module: chemical_reaction_rates_utilities
Description: Implements various useful utility functions for calculations of chemical abundances and rates.
Code lines: 19
Contained by: file chemical.reaction_rates.utilities.F90
Used by: function coldmodechemicalmasses, function coldmodecoldmodefraction, function simplechemicalmasses, function betaprofileradius, function betaprofileradiusgrowthrate, function isothermalradius, function simpleradius, function integrandluminosityxray, subroutine node_component_hot_halo_standard_formation, subroutine node_component_hot_halo_standard_outflow_return, subroutine node_component_hot_halo_standard_rate_compute

function: `chemicals_mass_to_density_conversion`

Description: Returns the conversion factor from mass of chemicals in ($M_\odot/M_{\rm atomic}$) to number density in cm^{-3} assuming that the mass is distributed uniformly in a sphere of the given `radius` (in Mpc).
Code lines: 9
Contained by: module `chemical_reaction_rates_utilities`
Modules used: `numerical_constants_astronomical`

file: `chemical.reaction_rates.zero.F90`

Description: An implementation of calculations of chemical reaction rates which assumes zero rates.
Code lines: 65

interface: `chemicalreactionratezero`

Description: Constructors for the `zero` chemical reaction rates class.
Code lines: 3
Contained by: file `chemical.reaction_rates.zero.F90`

function: `zeroconstructorparameters`

Description: Constructor for the `zero` chemical reaction rates class which takes a parameter set as input.
Code lines: 11
Contained by: file `chemical.reaction_rates.zero.F90`
Modules used: `input_parameters`

subroutine: `zerorates`

Description: Return zero rates of chemical reactions.
Code lines: 13
Contained by: file `chemical.reaction_rates.zero.F90`

file: `chemical.state.CIE_file.F90`

Description: Implements a chemical state class which reads and interpolates a collisional ionization equilibrium chemical state from a file.
Code lines: 635
Modules used: `fgsl`

interface: `chemicalstateciefile`

Description: Constructors for the "CIE file" chemical state class.
Code lines: 4
Contained by: file `chemical.state.CIE_file.F90`

subroutine: `ciefilechemicaldensities`

Description: Return the densities of chemical species at the given temperature and hydrogen density for the specified set of abundances and radiation field. Units of the returned electron density are cm$^-$3.
Code lines: 75
Contained by: file `chemical.state.CIE_file.F90`
Modules used: `abundances_structure` `chemical_abundances_structure` `iso_c_binding` `radiation_fields` `table_labels`

function: `ciefileconstructorinternal`

Description: Internal constructor for the "CIE file" chemical state class.
Code lines: 22
Contained by: file chemical.state.CIE_file.F90

function: ciefileconstructorparameters
Description: Constructor for the "CIE file" chemical state class which takes a parameter set as input.
Code lines: 19
Contained by: file chemical.state.CIE_file.F90
Modules used: galacticus_paths

subroutine: ciefiledestructor
Description: Destructor for the "CIE file" chemical state class.
Code lines: 10
Contained by: file chemical.state.CIE_file.F90
Modules used: numerical_interpolation

function: ciefileelectrondensity
Description: Return the electron density by interpolating in tabulated CIE data read from a file.
Code lines: 68
Contained by: file chemical.state.CIE_file.F90
Modules used: abundances_structure iso_c_binding radiation_fields table_labels

function: ciefileelectrondensitydensitylogslope
Description: Return the logarithmic slope of the electron density with respect to density assuming atomic CIE.
Code lines: 14
Contained by: file chemical.state.CIE_file.F90
Modules used: abundances_structure radiation_fields

function: ciefileelectrondensitytemperaturelogslope
Description: Return the logarithmic slope of the electron density with respect to temperature by interpolating in tabulated CIE data read from a file.
Code lines: 71
Contained by: file chemical.state.CIE_file.F90
Modules used: abundances_structure iso_c_binding radiation_fields table_labels

function: ciefileinterpolate
Description: Perform the interpolation.
Code lines: 14
Contained by: file chemical.state.CIE_file.F90
Modules used: iso_c_binding

subroutine: ciefileinterpolatingfactors
Description: Determine the interpolating paramters.
Code lines: 28
Contained by: file chemical.state.CIE_file.F90
Modules used: iso_c_binding numerical_interpolation

subroutine: `ciefilereadfile`
Description: Read in data from an chemical state file.
Code lines: 100
Contained by: file chemical.state.CIE_file.F90
Modules used: file_utilities, galacticus_display, galacticus_error, io_hdf5, iso_varying_string, table_labels

file: `chemical.state.F90`
Description: Contains a module that provides a class implementing the chemical state.
Code lines: 69

module: `chemical_states`
Description: Provides a class implementing the chemical state.
Code lines: 47
Contained by: file chemical.state.F90
Modules used: abundances_structure, chemical_abundances_structure, radiation_fields
Used by: file accretion.halo.simple.F90; file cooling.cooling_function.CMB_Compton.F90; function summationcoolingfunction; function summationcoolingfunctiondensitylogslope; file cooling.cooling_time.simple.F90; subroutine galacticus_function_classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; file nodes.property_extractor.ICM_SZ.F90; module node_component_hot_halo_standard; program test_cooling_functions

file: `chemical.state.atomic_CIE_cloudy.F90`
Description: Implements a chemical state class which utilizes the Cloudy code to compute state in collisional ionization equilibrium.
Code lines: 222

subroutine: `atomiccieclоudychemicaldensities`
Description: Return the densities of chemical species at the given temperature and hydrogen density for the specified set of abundances and radiation field. Units of the returned electron density are cm^{-}3.
Code lines: 13
Contained by: file chemical.state.atomic_CIE_cloudy.F90

function: `atomiccieclоudyconstructorinternal`
Description: Internal constructor for the "atomic CIE Cloudy" chemical state class.
Code lines: 19
Contained by: file chemical.state.atomic_CIE_cloudy.F90

function: `atomiccieclоudyconstructorparameters`
Description: Constructor for the "atomic CIE Cloudy" chemical state class which takes a parameter set as input.

Code lines: 10
Contained by: file chemical.state.atomic_CIE_cloudy.F90
Modules used: input_parameters

subroutine: atomiccieclоudydestructor
Description: Destructor for the "atomic CIE Cloudy" chemical state class.
Code lines: 9
Contained by: file chemical.state.atomic_CIE_cloudy.F90
Modules used: numerical_interpolation

function: atomiccieclоudyelectrondensity
Description: Return the electron density for collisional ionization equilibrium as computed by Cloudy.
Code lines: 12
Contained by: file chemical.state.atomic_CIE_cloudy.F90

function: atomiccieclоudyelectrondensitydensitylogslope
Description: Return the logarithmic slope of the electron density with respect to density for collisional ionization equilibrium as computed by Cloudy.
Code lines: 12
Contained by: file chemical.state.atomic_CIE_cloudy.F90

function: atomiccieclоudyelectrondensitytemperaturelogslope
Description: Return the logarithmic slope of the electron density with respect to temperature for collisional ionization equilibrium as computed by Cloudy. read from a file.
Code lines: 13
Contained by: file chemical.state.atomic_CIE_cloudy.F90

subroutine: atomiccieclоudytabulate
Description: Create the chemical state.
Code lines: 44
Contained by: file chemical.state.atomic_CIE_cloudy.F90
Modules used: galacticus_display, galacticus_paths, interfaces_cloudy_cie, string_handling, system_command

interface: chemicalstateatomiccieclоudy
Description: Constructors for the "atomic CIE Cloudy" chemical state class.
Code lines: 4
Contained by: file chemical.state.atomic_CIE_cloudy.F90

file: constant.F90
Code lines: 269

module: constants_nswc
Code lines: 265
Contained by: file constant.F90
Used by: module incomplete_gamma

function: depsln

Code lines: 13
Contained by: module constants_nswc

function: dpmpar
Code lines: 36
Contained by: module constants_nswc

function: dxparg
Code lines: 23
Contained by: module constants_nswc

function: epsln
Code lines: 13
Contained by: module constants_nswc

function: exparg
Code lines: 23
Contained by: module constants_nswc

function: ipmpar
Code lines: 83
Contained by: module constants_nswc

function: spmpar
Code lines: 36
Contained by: module constants_nswc

file: cooling.cold_mode.infall_rate.F90
Description: Contains a module that implements calculations of the infall rate from the cold mode.
Code lines: 41

module: cooling_cold_mode_infall_rates
Description: Implements calculations of the infall rate from the cold mode.
Code lines: 19
Contained by: file cooling.cold_mode.infall_rate.F90
Modules used: galacticus_nodes
Used by: subroutine galacticus_function_classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; file nodes.property_extractor.cold_mode_infall_rate.F90; module node_component_hot_halo_cold_mode

file: cooling.cold_mode.infall_rate.dynamical_time.F90
Description: Implementation of a calculation of cold mode infall rates assuming infall on a dynamical timescale.
Code lines: 104
Modules used: dark_matter_halo_scales

interface: coldmodeinfallratedynamicaltime

Description: Constructors for the dynamicalTime cooling time class.
Code lines: 4
Contained by: file cooling.cold_mode.infall_rate.dynamical_time.F90

function: dynamicaltimeconstructorinternal
Description: Internal constructor for the dynamical time cooling time class.
Code lines: 13
Contained by: file cooling.cold_mode.infall_rate.dynamical_time.F90
Modules used: galacticus_error galacticus_nodes

function: dynamicaltimeconstructorparameters
Description: Constructor for the dynamical time cooling time class which builds the object from a parameter set.
Code lines: 22
Contained by: file cooling.cold_mode.infall_rate.dynamical_time.F90
Modules used: input_parameters

subroutine: dynamicaltimedestructor
Description: Destructor for the dynamical time cold mode infall rate class.
Code lines: 7
Contained by: file cooling.cold_mode.infall_rate.dynamical_time.F90

function: dynamicaltimeinfallrate
Description: Computes the cold mode infall rate as a fraction of the halo dynamical time.
Code lines: 11
Contained by: file cooling.cold_mode.infall_rate.dynamical_time.F90
Modules used: galacticus_nodes

file: cooling.cooling_function.CIE_file.F90
Description: Implements a cooling function class which interpolates in a tabulated cooling function read from file.
Code lines: 542
Modules used: fgsl

function: ciefileconstructorinternal
Description: Internal constructor for the "CIE file" cooling function class.
Code lines: 17
Contained by: file cooling.cooling_function.CIE_file.F90

function: ciefileconstructorparameters
Description: Constructor for the "CIE file" cooling function class which takes a parameter set as input.
Code lines: 26
Contained by: file cooling.cooling_function.CIE_file.F90
Modules used: input_parameters

function: ciefilecoolingfunction
Description: Return the cooling function by interpolating in tabulated CIE data read from a file.
Code lines: 70
Contained by: file cooling.cooling_function.CIE_file.F90

Modules used: abundances_structure, chemical_abundances_structure, iso_c_binding, radiation_fields, table_labels

function: ciefilecoolingfunctiondensitylogslope
Description: Return the logarithmic slope of the cooling function with respect to density.
Code lines: 16
Contained by: file cooling.cooling_function.CIE_file.F90
Modules used: abundances_structure, chemical_abundances_structure, radiation_fields

function: ciefilecoolingfunctiontemperaturelogslope
Description: Return the slope of the cooling function with respect to temperature by interpolating in tabulated CIE data read from a file.
Code lines: 77
Contained by: file cooling.cooling_function.CIE_file.F90
Modules used: abundances_structure, chemical_abundances_structure, iso_c_binding, radiation_fields, table_labels

subroutine: ciefiledestructor
Description: Destructor for the "CIE file" cooling function class.
Code lines: 10
Contained by: file cooling.cooling_function.CIE_file.F90
Modules used: numerical_interpolation

function: ciefileinterpolate
Description: Perform the interpolation.
Code lines: 13
Contained by: file cooling.cooling_function.CIE_file.F90
Modules used: iso_c_binding

subroutine: ciefileinterpolatingfactors
Description: Determine the interpolating paramters.
Code lines: 28
Contained by: file cooling.cooling_function.CIE_file.F90
Modules used: iso_c_binding, numerical_interpolation

subroutine: ciefilereadfile
Description: Read in data from a cooling function file.
Code lines: 82
Contained by: file cooling.cooling_function.CIE_file.F90
Modules used: file_utilities, galacticus_display, galacticus_error, io_hdf5, iso_varying_string, table_labels

interface: coolingfunctionciefile
Description: Constructors for the "CIE file" cooling function class.
Code lines: 4

Contained by: file cooling.cooling_function.CIE_file.F90

file: cooling.cooling_function.CMB_Compton.F90
Description: Implements a cooling function class which implements cooling due to Compton scattering off of CMB photons.
Code lines: 172
Modules used: chemical_states

function: cmbcomptonconstructorinternal
Description: Internal constructor for the cmbCompton cooling function class.
Code lines: 9
Contained by: file cooling.cooling_function.CMB_Compton.F90
Modules used: input_parameters

function: cmbcomptonconstructorparameters
Description: Constructor for the "CMB Compton" cooling function class which takes a parameter set as input.
Code lines: 13
Contained by: file cooling.cooling_function.CMB_Compton.F90
Modules used: input_parameters

function: cmbcomptoncoolingfunction
Description: Return the cooling function due to Compton scattering off of CMB photons.
Code lines: 22
Contained by: file cooling.cooling_function.CMB_Compton.F90
Modules used: abundances_structure chemical_abundances_structure numerical_constants_physical numerical_constants_units radiation_fields

function: cmbcomptoncoolingfunctiondensitylogslope
Description: Return the logarithmic gradient with respect to density of the cooling function due to Compton scattering off of CMB photons.
Code lines: 17
Contained by: file cooling.cooling_function.CMB_Compton.F90
Modules used: abundances_structure chemical_abundances_structure radiation_fields

function: cmbcomptoncoolingfunctiontemperaturelogslope
Description: Return the logarithmic gradient with respect to temperature of the cooling function due to Compton scattering off of CMB photons.
Code lines: 20
Contained by: file cooling.cooling_function.CMB_Compton.F90
Modules used: abundances_structure chemical_abundances_structure radiation_fields

subroutine: cmbcomptondestructor
Description: Destructor for the cmbCompton cooling function class.
Code lines: 7
Contained by: file cooling.cooling_function.CMB_Compton.F90

function: `cmbcomptontemperature`
Description: Return the temperature of the cosmic microwave background.
Code lines: 26
Contained by: file `cooling.cooling_function.CMB_Compton.F90`
Modules used: `galacticus_error` `radiation_fields`

interface: `coolingfunctioncmbcompton`
Description: Constructors for the "CMB Compton" cooling function class.
Code lines: 4
Contained by: file `cooling.cooling_function.CMB_Compton.F90`

file: `cooling.cooling_function.F90`
Description: Contains a module that provides a class implementing cooling functions.
Code lines: 63

module: `cooling_functions`
Description: Provides a class implementing cooling functions.
Code lines: 41
Contained by: file `cooling.cooling_function.F90`
Modules used: `abundances_structure` `chemical_abundances_structure` `radiation_fields`
Used by: file `accretion.halo.cold_mode.F90`; file `cooling.cooling_time.simple.F90`; subroutine `galacticus_function_classes_destroy`; subroutine `galacticus_state_retrieve`; subroutine `galacticus_state_store`; file `nodes.property_extractor.ICM_Xray_luminosity.F90`; program `test_cooling_functions`

file: `cooling.cooling_function.atomic_CIE_Cloudy.F90`
Description: Implements a cooling function class which utilizes the Cloudy code to compute cooling in collisional ionization equilibrium.
Code lines: 206

function: `atomiccieclоudyconstructorinternal`
Description: Internal constructor for the "atomic CIE Cloudy" cooling function class.
Code lines: 14
Contained by: file `cooling.cooling_function.atomic_CIE_Cloudy.F90`

function: `atomiccieclоudyconstructorparameters`
Description: Constructor for the "atomic CIE Cloudy" cooling function class which takes a parameter set as input.
Code lines: 10
Contained by: file `cooling.cooling_function.atomic_CIE_Cloudy.F90`
Modules used: `input_parameters`

function: `atomiccieclоudycoolingfunction`
Description: Return the cooling function for collisional ionization equilibrium as computed by Cloudy.
Code lines: 13

Contained by: file cooling.cooling_function.atomic_CIE_Cloudy.F90

function: atomiccieclоudycoolingfunctiondensitylogslope
Description: Return the logarithmic slope of the cooling function with respect to density for collisional ionization equilibrium as computed by Cloudy.
Code lines: 13
Contained by: file cooling.cooling_function.atomic_CIE_Cloudy.F90

function: atomiccieclоudycoolingfunctiontemperaturelogslope
Description: Return the logarithmic slope of the cooling function with respect to temperature for collisional ionization equilibrium as computed by Cloudy. read from a file.
Code lines: 14
Contained by: file cooling.cooling_function.atomic_CIE_Cloudy.F90

subroutine: atomiccieclоudydestructor
Description: Destructor for the "atomic CIE Cloudy" cooling function class.
Code lines: 9
Contained by: file cooling.cooling_function.atomic_CIE_Cloudy.F90
Modules used: numerical_interpolation

subroutine: atomiccieclоudytabulate
Description: Create the chemical state.
Code lines: 44
Contained by: file cooling.cooling_function.atomic_CIE_Cloudy.F90
Modules used: galacticus_display galacticus_paths interfaces_cloudy_cie string_handling system_command

interface: coolingfunctionatomiccieclоudy
Description: Constructors for the "atomic CIE Cloudy" cooling function class.
Code lines: 4
Contained by: file cooling.cooling_function.atomic_CIE_Cloudy.F90

file: cooling.cooling_function.molecular_hydrogen_Galli_Palla.F90
Description: Implements a cooling function class which implements cooling from molecular hydrogen using the cooling function of Galli and Palla [1998].
Code lines: 392

interface: coolingfunctionmolecularhydrogengallipalla
Description: Constructors for the "molecular hydrogen (Galli & Palla)" cooling function class.
Code lines: 4
Contained by: file cooling.cooling_function.molecular_hydrogen_Galli_Palla.F90

subroutine: molecularhydrogengallipallacommonfactors
Description: Compute the ratio of critical number density to the hydrogen number density for use in molecular hydrogen cooling functions.
Code lines: 32
Contained by: file cooling.cooling_function.molecular_hydrogen_Galli_Palla.F90
Modules used: numerical_constants_prefixes

function: `molecularhydrogengallipallaconstructorinternal`
Description: Internal constructor for the "molecular hydrogen (Galli & Palla)" cooling function class.
Code lines: 19
Contained by: file `cooling.cooling_function.molecular_hydrogen_Galli_Palla.F90`

function: `molecularhydrogengallipallaconstructorparameters`
Description: Constructor for the "molecular hydrogen (Galli & Palla)" cooling function class which takes a parameter set as input.
Code lines: 10
Contained by: file `cooling.cooling_function.molecular_hydrogen_Galli_Palla.F90`
Modules used: `input_parameters`

function: `molecularhydrogengallipallacoolingfunction`
Description: Return the cooling function due to molecular hydrogen using the cooling function of Galli and Palla [1998] (which refers to the local thermodynamic equilibrium cooling function of Hollenbach and McKee [1979]). Cooling functions involving H_2^+ are computed using polynomial fits to the results of Suchkov and Shchekinov [1978] found by Andrew Benson by measuring curves from the original paper.
Code lines: 24
Contained by: file `cooling.cooling_function.molecular_hydrogen_Galli_Palla.F90`
Modules used: `abundances_structure` `chemical_abundances_structure` `radiation_fields`

function: `molecularhydrogengallipallacoolingfunctiondensitylogslope`
Description: Return the gradient with respect to density of the cooling function due to molecular hydrogen using the cooling function of Galli and Palla [1998].
Code lines: 43
Contained by: file `cooling.cooling_function.molecular_hydrogen_Galli_Palla.F90`
Modules used: `abundances_structure` `chemical_abundances_structure` `radiation_fields`

function: `molecularhydrogengallipallacoolingfunctionh2plus_electron`
Description: Compute the cooling function due to H_2^+–e^- interactions.
Code lines: 26
Contained by: file `cooling.cooling_function.molecular_hydrogen_Galli_Palla.F90`
Modules used: `chemical_abundances_structure`

function: `molecularhydrogengallipallacoolingfunctionh_h2`
Description: Compute the cooling function due to H–H_2 interactions.
Code lines: 22
Contained by: file `cooling.cooling_function.molecular_hydrogen_Galli_Palla.F90`
Modules used: `chemical_abundances_structure`

function: `molecularhydrogengallipallacoolingfunctionh_h2plus`
Description: Compute the cooling function due to H–H_2^+ interactions.
Code lines: 25
Contained by: file `cooling.cooling_function.molecular_hydrogen_Galli_Palla.F90`
Modules used: `chemical_abundances_structure`

function: `molecularhydrogengallipallacoolingfunctiontemperaturelogslope`
Description: Return the gradient with respect to temperature of the cooling function due to molecular hydrogen using the cooling function of Galli and Palla [1998].
Code lines: 75
Contained by: file `cooling.cooling_function.molecular_hydrogen_Galli_Palla.F90`
Modules used: `abundances_structure` `chemical_abundances_structure` `numerical_constants_prefixes` `radiation_fields`

file: `cooling.cooling_function.summation.F90`
Description: Implements a cooling function class which sums over other cooling functions.
Code lines: 253

type: `coolantlist`
Code lines: 3
Contained by: file `cooling.cooling_function.summation.F90`

interface: `coolingfunctionsummation`
Description: Constructors for the "summation" cooling function class.
Code lines: 4
Contained by: file `cooling.cooling_function.summation.F90`

function: `summationconstructorinternal`
Description: Internal constructor for the "summation" cooling function class.
Code lines: 14
Contained by: file `cooling.cooling_function.summation.F90`

function: `summationconstructorparameters`
Description: Constructor for the "summation" cooling function class which takes a parameter set as input.
Code lines: 21
Contained by: file `cooling.cooling_function.summation.F90`
Modules used: `input_parameters`

function: `summationcoolingfunction`
Description: Return the cooling function summed over other cooling functions.
Code lines: 21
Contained by: file `cooling.cooling_function.summation.F90`
Modules used: `abundances_structure` `chemical_abundances_structure` `chemical_states` `radiation_fields`

function: `summationcoolingfunctiondensitylogslope`
Description: Return the logarithmic gradient with respect to density of the cooling function due to Compton scattering off of CMB photons.
Code lines: 33
Contained by: file `cooling.cooling_function.summation.F90`
Modules used: `abundances_structure` `chemical_abundances_structure` `chemical_states` `galacticus_error` `radiation_fields`

function: summationcoolingfunctiontemperaturelogslope

Description: Return the logarithmic gradient with respect to temperature of the cooling function due to Compton scattering off of CMB photons.
Code lines: 32
Contained by: file cooling.cooling_function.summation.F90
Modules used: abundances_structure, chemical_abundances_structure, galacticus_error, radiation_fields

subroutine: summationdeepcopy

Description: Perform a deep copy for the summation cooling function class.
Code lines: 31
Contained by: file cooling.cooling_function.summation.F90
Modules used: galacticus_error

subroutine: summationdescriptor

Description: Add parameters to an input parameter list descriptor which could be used to recreate this object.
Code lines: 19
Contained by: file cooling.cooling_function.summation.F90
Modules used: fox_dom, input_parameters

subroutine: summationdestructor

Description: Destructor for the "summation" cooling function class.
Code lines: 16
Contained by: file cooling.cooling_function.summation.F90

file: cooling.cooling_radius.F90

Description: Contains a module that implements calculations of the cooling radius.
Code lines: 49

module: cooling_radii

Description: Provides a class that implements calculations of the cooling radius.
Code lines: 27
Contained by: file cooling.cooling_radius.F90
Modules used: galacticus_nodes
Used by: file cooling.infall_radius.cooling_and_freefall.F90; file cooling.infall_radius.cooling_radius.F90; subroutine galacticus_function_classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; file nodes.property_extractor.cooling_radius.F90; module node_component_black_hole_simple; module node_component_black_hole_standard

file: cooling.cooling_radius.beta_profile.F90

Description: Implementation of a cooling radius class for β-profile halos, assuming collisional ionization equilibrium such that cooling time scales as inverse density.
Code lines: 343
Modules used: cooling_times, cooling_times_available

cosmology_functions dark_matter_halo_scales
hot_halo_mass_distributions hot_halo_temperature_profiles
kind_numbers radiation_fields

subroutine: betaprofileautohook
Description: Attach to the calculation reset event.
Code lines: 8
Contained by: file cooling.cooling_radius.beta_profile.F90
Modules used: events_hooks

subroutine: betaprofilecalculationreset
Description: Reset the cooling radius calculation.
Code lines: 10
Contained by: file cooling.cooling_radius.beta_profile.F90

function: betaprofileconstructorinternal
Description: Internal constructor for the β-profile cooling radius class.
Code lines: 49
Contained by: file cooling.cooling_radius.beta_profile.F90
Modules used: abundances_structure array_utilities
chemical_abundances_structure galacticus_error
galacticus_nodes iso_varying_string
string_handling

function: betaprofileconstructorparameters
Description: Constructor for the β-profile cooling radius class which builds the object from a parameter set.
Code lines: 28
Contained by: file cooling.cooling_radius.beta_profile.F90
Modules used: input_parameters

subroutine: betaprofiledestructor
Description: Destructor for the β-profile cooling radius class.
Code lines: 15
Contained by: file cooling.cooling_radius.beta_profile.F90
Modules used: events_hooks

function: betaprofileradius
Description: Return the cooling radius in the β-profile model.
Code lines: 66
Contained by: file cooling.cooling_radius.beta_profile.F90
Modules used: abundances_structure chemical_abundances_structure
chemical_reaction_rates_utilities galacticus_nodes
hot_halo_mass_distributions

function: betaprofileradiusgrowthrate
Description: Returns the cooling radius growth rate (in Mpc/Gyr) in the hot atmosphere.
Code lines: 65
Contained by: file cooling.cooling_radius.beta_profile.F90

Modules used: abundances_structure, chemical_abundances_structure, chemical_reaction_rates_utilities, galacticus_nodes, hot_halo_mass_distributions

interface: `coolingradiusbetaprofile`
Description: Constructors for the betaProfile cooling radius class.
Code lines: 4
Contained by: file cooling.cooling_radius.beta_profile.F90

file: `cooling.cooling_radius.isothermal_profile.F90`
Description: Implementation of a cooling radius class for isothermal halos, assuming collisional ionization equilibrium such that cooling time scales as inverse density.
Code lines: 277
Modules used: cooling_times, cooling_times_available, cosmology_functions, dark_matter_halo_scales, hot_halo_mass_distributions, hot_halo_temperature_profiles, kind_numbers

interface: `coolingradiusisothermal`
Description: Constructors for the isothermal cooling radius class.
Code lines: 4
Contained by: file cooling.cooling_radius.isothermal_profile.F90

subroutine: `isothermalautohook`
Description: Attach to the calculation reset event.
Code lines: 8
Contained by: file cooling.cooling_radius.isothermal_profile.F90
Modules used: events_hooks

subroutine: `isothermalcalculationreset`
Description: Reset the cooling radius calculation.
Code lines: 10
Contained by: file cooling.cooling_radius.isothermal_profile.F90

function: `isothermalconstructorinternal`
Description: Internal constructor for the isothermal cooling radius class.
Code lines: 31
Contained by: file cooling.cooling_radius.isothermal_profile.F90
Modules used: abundances_structure, array_utilities, chemical_abundances_structure, galacticus_error, galacticus_nodes, iso_varying_string, string_handling

function: `isothermalconstructorparameters`
Description: Constructor for the isothermal cooling radius class which builds the object from a parameter set.
Code lines: 28
Contained by: file cooling.cooling_radius.isothermal_profile.F90
Modules used: input_parameters

subroutine: `isothermaldestructor`
Description: Destructor for the isothermal cooling radius class.
Code lines: 15
Contained by: file `cooling.cooling_radius.isothermal_profile.F90`
Modules used: `events_hooks`

function: `isothermalradius`
Description: Return the cooling radius in the isothermal model.
Code lines: 57
Contained by: file `cooling.cooling_radius.isothermal_profile.F90`
Modules used: `abundances_structure` `chemical_abundances_structure` `chemical_reaction_rates_utilities` `galacticus_nodes`

function: `isothermalradiusgrowthrate`
Description: Returns the cooling radius growth rate (in Mpc/Gyr) in the hot atmosphere.
Code lines: 30
Contained by: file `cooling.cooling_radius.isothermal_profile.F90`
Modules used: `dark_matter_halo_scales`

file: `cooling.cooling_radius.simple.F90`
Description: Implementation of a simple cooling radius class.
Code lines: 338
Modules used: `abundances_structure` `chemical_abundances_structure` `cooling_times` `cooling_times_available` `cosmology_functions` `hot_halo_mass_distributions` `hot_halo_temperature_profiles` `kind_numbers` `radiation_fields`

function: `coolingradiusroot`
Description: Root function which evaluates the difference between the cooling time at `radius` and the time available for cooling.
Code lines: 14
Contained by: file `cooling.cooling_radius.simple.F90`

interface: `coolingradiussimple`
Description: Constructors for the simple cooling radius class.
Code lines: 4
Contained by: file `cooling.cooling_radius.simple.F90`

subroutine: `simpleautohook`
Description: Attach to the calculation reset event.
Code lines: 8
Contained by: file `cooling.cooling_radius.simple.F90`
Modules used: `events_hooks`

subroutine: `simplecalculationreset`
Description: Reset the cooling radius calculation.
Code lines: 10

Contained by: file cooling.cooling_radius.simple.F90

function: simpleconstructorinternal
Description: Internal constructor for the simple cooling radius class.
Code lines: 30
Contained by: file cooling.cooling_radius.simple.F90
Modules used: abundances_structure, array_utilities, chemical_abundances_structure, galacticus_error, galacticus_nodes, iso_varying_string, string_handling

function: simpleconstructorparameters
Description: Constructor for the simple cooling radius class which builds the object from a parameter set.
Code lines: 25
Contained by: file cooling.cooling_radius.simple.F90
Modules used: input_parameters

subroutine: simpledestructor
Description: Destructor for the simple cooling radius class.
Code lines: 14
Contained by: file cooling.cooling_radius.simple.F90
Modules used: events_hooks

function: simpleradius
Description: Return the cooling radius in the simple model.
Code lines: 76
Contained by: file cooling.cooling_radius.simple.F90
Modules used: chemical_reaction_rates_utilities, galacticus_nodes, root_finder

function: simpleradiusgrowthrate
Description: Returns the cooling radius growth rate (in Mpc/Gyr) in the hot atmosphere.
Code lines: 53
Contained by: file cooling.cooling_radius.simple.F90
Modules used: abundances_structure, chemical_abundances_structure, galacticus_nodes

file: cooling.cooling_rate.Cole2000.F90
Description: Implementation of a cooling rate class for the Cole et al. [2000] cooling rate calculation.
Code lines: 123
Modules used: cooling_infall_radii, hot_halo_mass_distributions

function: cole2000constructorinternal
Description: Internal constructor for the Cole et al. [2000] cooling rate class.
Code lines: 9
Contained by: file cooling.cooling_rate.Cole2000.F90

function: cole2000constructorparameters

Description: Constructor for the Cole et al. [2000] cooling rate class which builds the object from a parameter set.
Code lines: 16
Contained by: file cooling.cooling_rate.Cole2000.F90
Modules used: input_parameters

subroutine: cole2000destructor
Description: Destructor for the Cole et al. [2000] cooling rate class.
Code lines: 8
Contained by: file cooling.cooling_rate.Cole2000.F90

function: cole2000rate
Description: Returns the cooling rate (in $M_\odot$ Gyr^{-1}) in the hot atmosphere for the White and Frenk [1991] cooling rate model.
Code lines: 37
Contained by: file cooling.cooling_rate.Cole2000.F90
Modules used: galacticus_nodes numerical_constants_math

interface: coolingratecole2000
Description: Constructors for the Cole et al. [2000] cooling rate class.
Code lines: 4
Contained by: file cooling.cooling_rate.Cole2000.F90

file: cooling.cooling_rate.F90
Description: Contains a module that implements calculations of the cooling rate.
Code lines: 40

module: cooling_rates
Description: Provides a class that implements calculations of the cooling rate.
Code lines: 18
Contained by: file cooling.cooling_rate.F90
Modules used: galacticus_nodes
Used by: subroutine galacticus_function_classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; file nodes.property_extractor.cooling_rate.F90; module node_component_hot_halo_standard; module node_component_hot_halo_very_simple; module node_component_mass_flow_statistics_standard

file: cooling.cooling_rate.White-Frenk.F90
Description: Implementation of a cooling rate class for the White and Frenk [1991] cooling rate calculation.
Code lines: 143
Modules used: cooling_infall_radii dark_matter_halo_scales hot_halo_mass_distributions

interface: coolingratewhitefrenk1991

Description: Constructors for the White and Frenk [1991] cooling rate class.
Code lines: 4
Contained by: file cooling.cooling_rate.White-Frenk.F90

function: whitefrenk1991constructorinternal
Description: Internal constructor for the White and Frenk [1991] cooling rate class.
Code lines: 16
Contained by: file cooling.cooling_rate.White-Frenk.F90
Modules used: array_utilities galacticus_error galacticus_nodes

function: whitefrenk1991constructorparameters
Description: Constructor for the White and Frenk [1991] cooling rate class which builds the object from a parameter set.
Code lines: 28
Contained by: file cooling.cooling_rate.White-Frenk.F90
Modules used: input_parameters

subroutine: whitefrenk1991destructor
Description: Destructor for the White and Frenk [1991] cooling rate class.
Code lines: 9
Contained by: file cooling.cooling_rate.White-Frenk.F90

function: whitefrenk1991rate
Description: Returns the cooling rate (in $M_\odot$ Gyr^{-1}) in the hot atmosphere for the White and Frenk [1991] cooling rate model.
Code lines: 35
Contained by: file cooling.cooling_rate.White-Frenk.F90
Modules used: galacticus_nodes numerical_constants_math

file: cooling.cooling_rate.cut_off.F90
Description: Implementation of a cooling rate class which modifies another cooling rate by cutting off cooling above some virial velocity.
Code lines: 168
Modules used: cosmology_functions dark_matter_halo_scales

interface: coolingratecutoff
Description: Constructors for the cut off cooling rate class.
Code lines: 4
Contained by: file cooling.cooling_rate.cut_off.F90

function: cutoffconstructorinternal
Description: Internal constructor for the cut off cooling rate class.
Code lines: 15
Contained by: file cooling.cooling_rate.cut_off.F90
Modules used: galacticus_error

function: cutoffconstructorparameters
Description: Constructor for the cut off cooling rate class which builds the object from a parameter set.
Code lines: 54

Contained by: file cooling.cooling_rate.cut_off.F90
Modules used: input_parameters

subroutine: cutoffdestructor
Description: Destructor for the cut off cooling rate class.
Code lines: 8
Contained by: file cooling.cooling_rate.cut_off.F90

function: cutoffrate
Description: Returns the cooling rate (in $M_\odot$ Gyr^{-1}) in the hot atmosphere for a model in which this rate is cut off before/after a given epoch and below a given virial velocity.
Code lines: 24
Contained by: file cooling.cooling_rate.cut_off.F90
Modules used: galacticus_nodes

file: cooling.cooling_rate.multiplier.F90
Description: Implementation of a cooling rate class which modifies another cooling rate by cutting off cooling in satellites.
Code lines: 97

interface: coolingratemultiplier
Description: Constructors for the cut off cooling rate class.
Code lines: 4
Contained by: file cooling.cooling_rate.multiplier.F90

function: multiplierconstructorinternal
Description: Internal constructor for the cut off cooling rate class.
Code lines: 8
Contained by: file cooling.cooling_rate.multiplier.F90

function: multiplierconstructorparameters
Description: Constructor for the cut off cooling rate class which builds the object from a parameter set.
Code lines: 22
Contained by: file cooling.cooling_rate.multiplier.F90
Modules used: input_parameters

subroutine: multiplierdestructor
Description: Destructor for the cut off cooling rate class.
Code lines: 7
Contained by: file cooling.cooling_rate.multiplier.F90

function: multiplierrate
Description: Returns the cooling rate (in $M_\odot$ Gyr^{-1}) in the hot atmosphere for a model in which this rate is multiplied by some fixed value.
Code lines: 9
Contained by: file cooling.cooling_rate.multiplier.F90

file: cooling.cooling_rate.no_cooling_satellites.F90
Description: Implementation of a cooling rate class which modifies another cooling rate by cutting off cooling in satellites.

Code lines: 91

interface: `coolingratenocoolingsatellites`
Description: Constructors for the cut off cooling rate class.
Code lines: 4
Contained by: file `cooling.cooling_rate.no_cooling_satellites.F90`

function: `nocoolingsatellitesconstructorinternal`
Description: Internal constructor for the cut off cooling rate class.
Code lines: 8
Contained by: file `cooling.cooling_rate.no_cooling_satellites.F90`
Modules used: `array_utilities`

function: `nocoolingsatellitesconstructorparameters`
Description: Constructor for the cut off cooling rate class which builds the object from a parameter set.
Code lines: 13
Contained by: file `cooling.cooling_rate.no_cooling_satellites.F90`
Modules used: `input_parameters`

subroutine: `nocoolingsatellitesdestructor`
Description: Destructor for the cut off cooling rate class.
Code lines: 7
Contained by: file `cooling.cooling_rate.no_cooling_satellites.F90`

function: `nocoolingsatellitesrate`
Description: Returns the cooling rate (in $M_\odot$ Gyr^{-1}) in the hot atmosphere for a model in which this rate is cut off in satellites.
Code lines: 13
Contained by: file `cooling.cooling_rate.no_cooling_satellites.F90`

file: `cooling.cooling_rate.simple.F90`
Description: Implementation of a simple cooling rate class.
Code lines: 85

interface: `coolingratesimple`
Description: Constructors for the simple cooling rate class.
Code lines: 4
Contained by: file `cooling.cooling_rate.simple.F90`

function: `simpleconstructorinternal`
Description: Internal constructor for the simple cooling rate class.
Code lines: 8
Contained by: file `cooling.cooling_rate.simple.F90`

function: `simpleconstructorparameters`
Description: Constructor for the simple cooling rate class which builds the object from a parameter set.
Code lines: 20
Contained by: file `cooling.cooling_rate.simple.F90`
Modules used: `input_parameters`

function: `simplerate`
Description: Returns the cooling rate (in $M_\odot$ Gyr^{-1}) in the hot atmosphere for a model in which this rate is always simple.
Code lines: 11
Contained by: file `cooling.cooling_rate.simple.F90`
Modules used: `galacticus_nodes`

file: `cooling.cooling_rate.simple_scaling.F90`
Description: Implementation of a cooling rate class in which the cooling rate scales with the mass of the halo.
Code lines: 162
Modules used: `cosmology_functions`

interface: `coolingratesimplescaling`
Description: Constructors for the simple caling cooling rate class.
Code lines: 4
Contained by: file `cooling.cooling_rate.simple_scaling.F90`

function: `simplescalingconstructorinternal`
Description: Internal constructor for the simple caling cooling rate class.
Code lines: 19
Contained by: file `cooling.cooling_rate.simple_scaling.F90`
Modules used: `array_utilities` `galacticus_error` `galacticus_nodes`

function: `simplescalingconstructorparameters`
Description: Constructor for the simple caling cooling rate class which builds the object from a parameter set.
Code lines: 54
Contained by: file `cooling.cooling_rate.simple_scaling.F90`
Modules used: `input_parameters`

subroutine: `simplescalingdestructor`
Description: Destructor for the simple caling cooling rate class.
Code lines: 7
Contained by: file `cooling.cooling_rate.simple_scaling.F90`

function: `simplescalingrate`
Description: Returns the cooling rate (in $M_\odot$ Gyr^{-1}) in the hot atmosphere for a model in which this rate scales with the mass of the halo.
Code lines: 28
Contained by: file `cooling.cooling_rate.simple_scaling.F90`
Modules used: `galacticus_nodes`

file: `cooling.cooling_rate.velocity_maximum_scaling.F90`
Description: Implementation of a cooling rate class in which the cooling rate scales with the peak circular velocity in the halo.
Code lines: 206
Modules used: `cosmology_functions` `dark_matter_profiles_dmo`

math_exponentiation

interface: `coolingratevelocitymaximumscaling`
Description: Constructors for the velocity maximum scaling cooling rate class.
Code lines: 4
Contained by: file cooling.cooling_rate.velocity_maximum_scaling.F90

function: `velocitymaximumscalingconstructorinternal`
Description: Internal constructor for the velocity maximum scaling cooling rate class.
Code lines: 27
Contained by: file cooling.cooling_rate.velocity_maximum_scaling.F90
Modules used: array_utilities galacticus_error galacticus_nodes

function: `velocitymaximumscalingconstructorparameters`
Description: Constructor for the velocity maximum scaling cooling rate class which builds the object from a parameter set.
Code lines: 81
Contained by: file cooling.cooling_rate.velocity_maximum_scaling.F90
Modules used: input_parameters

subroutine: `velocitymaximumscalingdestructor`
Description: Destructor for the velocity maximum scaling cooling rate class.
Code lines: 8
Contained by: file cooling.cooling_rate.velocity_maximum_scaling.F90

function: `velocitymaximumscalingrate`
Description: Returns the cooling rate (in $M_\odot$ Gyr^{-1}) in the hot atmosphere for a model in which this rate scales with the maximum circular velocity of the halo.
Code lines: 30
Contained by: file cooling.cooling_rate.velocity_maximum_scaling.F90
Modules used: galacticus_nodes

file: `cooling.cooling_rate.zero.F90`
Description: Implementation of a zero cooling rate class.
Code lines: 60

interface: `coolingratezero`
Description: Constructors for the zero cooling rate class.
Code lines: 3
Contained by: file cooling.cooling_rate.zero.F90

function: `zeroconstructorparameters`
Description: Constructor for the zero cooling rate class which builds the object from a parameter set.
Code lines: 10
Contained by: file cooling.cooling_rate.zero.F90
Modules used: input_parameters

function: `zerorate`

Description: Returns the cooling rate (in $M_\odot$ Gyr^{-1}) in the hot atmosphere for a model in which this rate is always zero.
Code lines: 9
Contained by: file cooling.cooling_rate.zero.F90

file: cooling.cooling_time.F90
Description: Contains a module that implements calculations of the cooling time.
Code lines: 64

module: cooling_times
Description: Implements calculations of the cooling time.
Code lines: 42
Contained by: file cooling.cooling_time.F90
Modules used: abundances_structure, chemical_abundances_structure, radiation_fields
Used by: file cooling.cooling_radius.beta_profile.F90; file cooling.cooling_radius.isothermal_profile.F90; file cooling.cooling_radius.simple.F90; subroutine galacticus_function_classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store

file: cooling.cooling_time.simple.F90
Description: Implementation of a simple cooling time class.
Code lines: 156
Modules used: chemical_states, cooling_functions

interface: coolingtimesimple
Description: Constructors for the simple cooling time class.
Code lines: 4
Contained by: file cooling.cooling_time.simple.F90

function: simpleconstructorinternal
Description: Internal constructor for the simple cooling time class.
Code lines: 10
Contained by: file cooling.cooling_time.simple.F90

function: simpleconstructorparameters
Description: Constructor for the simple cooling time class which builds the object from a parameter set.
Code lines: 25
Contained by: file cooling.cooling_time.simple.F90
Modules used: input_parameters

subroutine: simpledestructor
Description: Destructor for the simple cooling time class.
Code lines: 8
Contained by: file cooling.cooling_time.simple.F90

function: simplegradientdensitylogarithmic
Description: Return $\mathrm{d}\ln t_{\rm cool}/\mathrm{d}\ln\rho$ for gas at the given `temperature` (in Kelvin), `density` (in $M_\odot$ Mpc^{-3}), composition specified by `gasAbundances` and experiencing a radiation field as described by `radiation`.

Code lines: 12
Contained by: file cooling.cooling_time.simple.F90

function: simplegradienttemperaturelogarithmic
Description: Return $\mathrm{d}\ln t_{\rm cool}/\mathrm{d}\ln T$ for gas at the given temperature (in Kelvin), density (in $M_\odot$ Mpc^{-3}), composition specified by gasAbundances and experiencing a radiation field as described by radiation.
Code lines: 12
Contained by: file cooling.cooling_time.simple.F90

function: simpletime
Description: Compute the cooling time (in Gyr) for gas at the given temperature (in Kelvin), density (in $M_\odot$ Mpc^{-3}), composition specified by gasAbundances and experiencing a radiation field as described by radiation.
Code lines: 30
Contained by: file cooling.cooling_time.simple.F90
Modules used: numerical_constants_astronomical numerical_constants_physical

file: cooling.freefall_radii.F90
Description: Contains a module that implements calculations of the freefall radius.
Code lines: 48

module: freefall_radii
Description: Provides a class that implements calculations of the freefall radius.
Code lines: 26
Contained by: file cooling.freefall_radii.F90
Modules used: galacticus_nodes
Used by: file cooling.infall_radius.cooling_-and_freefall.F90 subroutine galacticus_function_-classes_destroy
subroutine galacticus_state_retrieve subroutine galacticus_state_store

file: cooling.freefall_radii.dark_matter_halo.F90
Description: Implementation of a simple freefall radius class.
Code lines: 116
Modules used: cooling_freefall_times_available dark_matter_profiles_dmo

function: darkmatterhaloconstructorinternal
Description: Internal constructor for the darkMatterHalo freefall radius class.
Code lines: 9
Contained by: file cooling.freefall_radii.dark_matter_halo.F90

function: darkmatterhaloconstructorparameters
Description: Constructor for the darkMatterHalo freefall radius class which builds the object from a parameter set.
Code lines: 16
Contained by: file cooling.freefall_radii.dark_matter_halo.F90
Modules used: input_parameters

subroutine: darkmatterhalodestructor

Description: Destructor for the darkMatterHalo freefall radius class.
Code lines: 8
Contained by: file cooling.freefall_radii.dark_matter_halo.F90

function: darkmatterhaloradius
Description: Return the freefall radius in the darkMatterHalo model.
Code lines: 12
Contained by: file cooling.freefall_radii.dark_matter_halo.F90

function: darkmatterhaloradiusgrowthrate
Description: Returns the freefall radius growth rate (in Mpc/Gyr) in the hot atmosphere.
Code lines: 14
Contained by: file cooling.freefall_radii.dark_matter_halo.F90

interface: freefallradiusdarkmatterhalo
Description: Constructors for the darkMatterHalo freefall radius class.
Code lines: 4
Contained by: file cooling.freefall_radii.dark_matter_halo.F90

file: cooling.freefall_time_available.F90
Description: Contains a module that implements calculations of the time available for freefall in cooling calculations.
Code lines: 48

module: cooling_freefall_times_available
Description: Provides a class that implements calculations of the freefall radius.
Code lines: 26
Contained by: file cooling.freefall_time_available.F90
Modules used: galacticus_nodes
Used by: file cooling.freefall_radii.dark_matter_halo.F90; subroutine galacticus_function_classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store

file: cooling.freefall_time_available.halo_formation.F90
Description: Implementation of the Cole et al. [2000] method for computing the time available for freefall in cooling calculations in hot halos.
Code lines: 88

interface: freefalltimeavailablehaloformation
Description: Constructors for the haloFormation freefall time available class.
Code lines: 3
Contained by: file cooling.freefall_time_available.halo_formation.F90

function: haloformationconstructorparameters
Description: Constructor for the haloFormation freefall time available class which builds the object from a parameter set.
Code lines: 15
Contained by: file cooling.freefall_time_available.halo_formation.F90
Modules used: galacticus_error; galacticus_nodes; input_parameters

function: `haloformationtimeavailable`
Description: Compute the time available for freefall using the Cole et al. [2000] method. Specifically, the time available is assumed to be the time since the halo formation event.
Code lines: 15
Contained by: file `cooling.freefall_time_available.halo_formation.F90`
Modules used: `galacticus_nodes`

function: `haloformationtimeavailableincreaserate`
Description: Compute the rate of increase of the time available for freefall using the Cole et al. [2000] method. We return a rate of 1.
Code lines: 11
Contained by: file `cooling.freefall_time_available.halo_formation.F90`

file: `cooling.infall_radius.F90`
Description: Contains a module that implements calculations of the infall radius for cooling calculations.
Code lines: 47

module: `cooling_infall_radii`
Description: Provides a class that implements calculations of the infall radius for cooling calculations.
Code lines: 25
Contained by: file `cooling.infall_radius.F90`
Modules used: `galacticus_nodes`
Used by: file `cooling.cooling_rate.Cole2000.F90`; file `cooling.cooling_rate.White-Frenk.F90`; subroutine `galacticus_function_classes_destroy`; subroutine `galacticus_state_retrieve`; subroutine `galacticus_state_store`; module `node_component_hot_halo_standard`

file: `cooling.infall_radius.cooling_and_freefall.F90`
Description: Implementation of an infall radius calculation in which the infall radius is the smaller of the cooling and freefall radii.
Code lines: 115
Modules used: `cooling_radii`; `freefall_radii`

function: `coolingfreefallconstructorinternal`
Description: Internal constructor for the cooling radius infall radii class.
Code lines: 10
Contained by: file `cooling.infall_radius.cooling_and_freefall.F90`
Modules used: `galacticus_error`

function: `coolingfreefallconstructorparameters`
Description: Constructor for the cooling radius infall radii class which builds the object from a parameter set.
Code lines: 16
Contained by: file `cooling.infall_radius.cooling_and_freefall.F90`
Modules used: `input_parameters`

subroutine: `coolingfreefalldestructor`

Description: Destructor for the cooling radius infall radii class.
Code lines: 8
Contained by: file cooling.infall_radius.cooling_and_freefall.F90

function: coolingfreefallradius
Description: Return the infall radius in the "cooling radius" model in Mpc/Gyr.
Code lines: 11
Contained by: file cooling.infall_radius.cooling_and_freefall.F90

function: coolingfreefallradiusincreaserate
Description: Return the growth rate of the infall radius in the "cooling radius" model in Mpc/Gyr.
Code lines: 15
Contained by: file cooling.infall_radius.cooling_and_freefall.F90

interface: coolinginfallradiuscoolingfreefall
Description: Constructors for the cooling radius infall radii class.
Code lines: 4
Contained by: file cooling.infall_radius.cooling_and_freefall.F90

file: cooling.infall_radius.cooling_radius.F90
Description: Implementation of a simple infall radius calculation, simply assuming that the infall radius equals the cooling radius.
Code lines: 98
Modules used: cooling_radii

interface: coolinginfallradiuscoolingradius
Description: Constructors for the cooling radius infall radii class.
Code lines: 4
Contained by: file cooling.infall_radius.cooling_radius.F90

function: coolingradiusconstructorinternal
Description: Internal constructor for the cooling radius infall radii class.
Code lines: 9
Contained by: file cooling.infall_radius.cooling_radius.F90
Modules used: galacticus_error

function: coolingradiusconstructorparameters
Description: Constructor for the cooling radius infall radii class which builds the object from a parameter set.
Code lines: 13
Contained by: file cooling.infall_radius.cooling_radius.F90
Modules used: input_parameters

subroutine: coolingradiusdestructor
Description: Destructor for the cooling radius infall radii class.
Code lines: 7
Contained by: file cooling.infall_radius.cooling_radius.F90

function: coolingradiusradius

Description: Return the infall radius in the "cooling radius" model in Mpc/Gyr.
Code lines: 8
Contained by: file cooling.infall_radius.cooling_radius.F90

function: coolingradiusradiusincreaserate
Description: Return the growth rate of the infall radius in the "cooling radius" model in Mpc/Gyr.
Code lines: 8
Contained by: file cooling.infall_radius.cooling_radius.F90

file: cooling.specific_angular_momentum.F90
Description: Contains a module that implements calculations of the specific angular momentum of cooling gas.
Code lines: 43

module: cooling_specific_angular_momenta
Description: Provides a class that implements calculations of the specific angular momentum of cooling gas.
Code lines: 21
Contained by: file cooling.specific_angular_momentum.F90
Modules used: galacticus_nodes
Used by: subroutine galacticus_function_-classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; module node_component_hot_halo_-standard

file: cooling.specific_angular_momentum.constant_rotation.F90
Description: Implementation of a specific angular momentum of cooling gas class assuming a constant rotation velocity as a function of radius.
Code lines: 236
Modules used: dark_matter_profiles_dmo; hot_halo_mass_distributions; kind_numbers

function: constantrotationangularmomentumspecific
Description: Return the specific angular momentum of cooling gas in the constantRotation model.
Code lines: 63
Contained by: file cooling.specific_angular_momentum.constant_rotation.F90
Modules used: galacticus_error; galacticus_nodes; numerical_constants_physical

subroutine: constantrotationautohook
Description: Attach to the calculation reset event.
Code lines: 8
Contained by: file cooling.specific_angular_momentum.constant_rotation.F90
Modules used: events_hooks

subroutine: constantrotationcalculationreset
Description: Reset the specific angular momentum of cooling gas calculation.
Code lines: 9
Contained by: file cooling.specific_angular_momentum.constant_rotation.F90

function: `constantrotationconstructorinternal`
Description: Internal constructor for the darkMatterHalo freefall radius class.
Code lines: 13
Contained by: file `cooling.specific_angular_momentum.constant_rotation.F90`

function: `constantrotationconstructorparameters`
Description: Constructor for the constantRotation freefall radius class which builds the object from a parameter set.
Code lines: 48
Contained by: file `cooling.specific_angular_momentum.constant_rotation.F90`
Modules used: `input_parameters`

subroutine: `constantrotationdestructor`
Description: Destructor for the constant rotation specific angular momentum of cooling gas class.
Code lines: 10
Contained by: file `cooling.specific_angular_momentum.constant_rotation.F90`
Modules used: `events_hooks`

interface: `coolingspecificangularmomentumconstantrotation`
Description: Constructors for the constantRotation specific angular momentum of cooling gas class.
Code lines: 4
Contained by: file `cooling.specific_angular_momentum.constant_rotation.F90`

file: `cooling.specific_angular_momentum.mean.F90`
Description: Implementation of a specific angular momentum of cooling gas class in which all gas has the mean specific angular momentum of the hot gas halo.
Code lines: 68

interface: `coolingspecificangularmomentummean`
Description: Constructors for the mean specific angular momentum of cooling gas class.
Code lines: 3
Contained by: file `cooling.specific_angular_momentum.mean.F90`

function: `meanangularmomentumspecific`
Description: Return the specific angular momentum of cooling gas in the mean model.
Code lines: 14
Contained by: file `cooling.specific_angular_momentum.mean.F90`
Modules used: `galacticus_nodes`

function: `meanconstructorparameters`
Description: Constructor for the mean freefall radius class which builds the object from a parameter set.
Code lines: 10
Contained by: file `cooling.specific_angular_momentum.mean.F90`
Modules used: `input_parameters`

file: `cooling.time_available.F90`
Description: Contains a module that implements calculations of the time available for cooling.
Code lines: 46

module: `cooling_times_available`
Description: Provides a class that implements calculations of the time available for cooling.
Code lines: 24
Contained by: file `cooling.time_available.F90`
Modules used: `galacticus_nodes`
Used by: file `cooling.cooling_radius.beta_profile.F90`; file `cooling.cooling_radius.isothermal_profile.F90`; file `cooling.cooling_radius.simple.F90`; subroutine `galacticus_function_classes_destroy`; subroutine `galacticus_state_retrieve`; subroutine `galacticus_state_store`

file: `cooling.time_available.White-Frenk.F90`
Description: Implementation of the White and Frenk [1991] time available for cooling class.
Code lines: 127
Modules used: `dark_matter_halo_scales`

interface: `coolingtimeavailablewhitefrenk1991`
Description: Constructors for the White and Frenk [1991] time available for cooling class.
Code lines: 4
Contained by: file `cooling.time_available.White-Frenk.F90`

function: `whitefrenk1991constructorinternal`
Description: Internal constructor for the White and Frenk [1991] cooling rate class.
Code lines: 11
Contained by: file `cooling.time_available.White-Frenk.F90`
Modules used: `galacticus_error`

function: `whitefrenk1991constructorparameters`
Description: Constructor for the White and Frenk [1991] time available for cooling class which builds the object from a parameter set.
Code lines: 22
Contained by: file `cooling.time_available.White-Frenk.F90`
Modules used: `input_parameters`

subroutine: `whitefrenk1991destructor`
Description: Destructor for the White and Frenk [1991] cooling rate class.
Code lines: 7
Contained by: file `cooling.time_available.White-Frenk.F90`

function: `whitefrenk1991timeavailable`
Description: Returns the time available for cooling (in units of Gyr) in the hot atmosphere for the White and Frenk [1991] model.
Code lines: 22
Contained by: file `cooling.time_available.White-Frenk.F90`
Modules used: `galacticus_nodes`

function: `whitefrenk1991timeavailableincreaserate`
Description: Compute the rate of increase of the time available for cooling using the White and Frenk [1991] method. We return a rate of 1, even though technically it can depend on halo properties.

Code lines: 11
Contained by: file cooling.time_available.White-Frenk.F90

file: cooling.time_available.halo_formation.F90
Description: Implementation of the Cole et al. [2000] time available for cooling class.
Code lines: 84

interface: coolingtimeavailableformationtime
Description: Constructors for the Cole et al. [2000] time available for cooling class.
Code lines: 3
Contained by: file cooling.time_available.halo_formation.F90

function: formationtimeconstructorparameters
Description: Constructor for the Cole et al. [2000] time available for cooling class which builds the object from a parameter set.
Code lines: 14
Contained by: file cooling.time_available.halo_formation.F90
Modules used: galacticus_error galacticus_nodes input_parameters

function: formationtimetimeavailable
Description: Returns the time available for cooling (in units of Gyr) in the hot atmosphere for the Cole et al. [2000] model.
Code lines: 14
Contained by: file cooling.time_available.halo_formation.F90
Modules used: galacticus_nodes

function: formationtimetimeavailableincreaserate
Description: Compute the rate of increase of the time available for cooling using the Cole et al. [2000] method. We return a rate of 1, even though technically it can depend on halo properties.
Code lines: 11
Contained by: file cooling.time_available.halo_formation.F90

file: cosmology.functions.F90
Description: Contains a module which provides an object that implements cosmological functions.
Code lines: 249

module: cosmology_functions
Description: Provides an object that implements cosmological functions.
Code lines: 227
Contained by: file cosmology.functions.F90
Used by: file accretion.halo.simple.F90 program benchmark_stellar_populations_luminosities file cooling.cooling_radius.beta_profile.F90 file cooling.cooling_radius.isothermal_profile.F90 file cooling.cooling_radius.simple.F90 file cooling.cooling_rate.cut_off.F90 file cooling.cooling_rate.simple_scaling.F90 file cooling.cooling_rate.velocity_maximum_scaling.F90

file dark_matter_halos.mass_accretion_history.Correa2015.F90
file dark_matter_halos.mass_accretion_history.Wechsler2002.F90
file dark_matter_halos.mass_loss_rates.vanDenBosch.F90
file dark_matter_halos.scales.virial_density_contrast.F90
file dark_matter_halos.spins.distributions.N-body_errors.F90
file dark_matter_profiles.structure.concentration.Bullock2001.F90
file dark_matter_profiles.structure.concentration.Correa2015.F90
file dark_matter_profiles.structure.concentration.Diemer-Kravtsov2014.F90
file dark_matter_profiles.structure.concentration.Dutton-Macciò2014.F90
file dark_matter_profiles.structure.concentration.Gao2008.F90
file dark_matter_profiles.structure.concentration.Klypin2015.F90
file dark_matter_profiles.structure.concentration.Ludlow2016_fit.F90
file dark_matter_profiles.structure.concentration.MunozCuartas2011.F90
file dark_matter_profiles.structure.concentration.NFW.F90
file dark_matter_profiles.structure.concentration.Prada2012.F90
file dark_matter_profiles.structure.concentration.Schneider2015.F90
file dark_matter_profiles.structure.concentration.Zhao2009.F90
function dark_matter_profile_mass_definition
file dark_matter_profiles.structure.scale.Ludlow2016.F90
file dark_matter_profiles.structure.scale.concentration.F90
file galactic.filters.stellar_apparent_magnitudes.F90
function galactic_structure_radius_enclosing_density
subroutine galacticus_function_classes_destroy
function hivshalomassrelationpadmanabhan2017constructorinternal
function hivshalomassrelationpadmanabhan2017constructorparameters
function blackholebulgerelationconstructorinternal
function blackholebulgerelationconstructorparameters
file galacticus.output.analyses.color_distribution.SDSS.F90
function concentrationdistributioncdmcococonstructorinternal
function concentrationdistributioncdmcococonstructorparameters
function concentrationvshalomasscdmludlow2016constructorinternal
function concentrationvshalomasscdmludlow2016constructorparameters
file galacticus.output.analyses.correlation_function.F90
function correlationfunctionconstructorfile
file galacticus.output.analyses.galaxy_sizes_SDSS.F90
file galacticus.output.analyses.luminosity_function.F90
file galacticus.output.analyses.luminosity_function.Halpha.F90
file galacticus.output.analyses.mass_function_HI.F90

file galacticus.output.analyses.mass_-function_stellar.F90
function massmetallicityandrews2013constructorparameters
function massmetallicityblanc2017constructorparameters
function morphologicalfractiongamamoffett2016constructorparameters
file galacticus.output.analyses.property_-operator.cosmology.luminosity_-distance.F90
function stellarvshalomassrelationleauthaud2012constructorparameters
function output_analysis_output_-weight_survey_volume
subroutine galacticus_extra_output_-halo_fourier_profile
subroutine galacticus_state_store
file geometry.surveys.Baldry-2012-GAMA.F90
file geometry.surveys.Caputi-2011-UKIDSS-UDS.F90
function davidzon2013vipersconstructorinternal
file geometry.surveys.Hearin-2014-SDSS.F90
function liwhite2009sdssconstructorinternal
file geometry.surveys.Montero-Dorta-2009-SDSS.F90
function moustakas2013primusconstructorinternal
file geometry.surveys.Tomczak-2014-ZFOURGE.F90
subroutine halo_model_projected_-correlation
file hot_halo.outflow_-reincorporation.velocity_maximum_-scaling.F90
function massmetallicityandrews2013constructorinternal
function massmetallicityblanc2017constructorinternal
function morphologicalfractiongamamoffett2016constructorinternal
file galacticus.output.analyses.property_-operator.cosmology.angular_-distance.F90
file galacticus.output.analyses.spin_-distribution.Bett2007.F90
function stellarvshalomassrelationleauthaud2012constructorparameters
file galacticus.output.analyses.weight_-operator.cosmology.volume.F90
subroutine galacticus_state_retrieve
file geometry.lightcones.square.F90
function baldry2012gamaconstructorparameters
file geometry.surveys.Davidzon-2013-VIPERS.F90
file geometry.surveys.Gunawardhana-2013-SDSS.F90
file geometry.surveys.Li-White-2009-SDSS.F90
subroutine martin2010alfalfarandomsinitialize
file geometry.surveys.Moustakas-2013-PRIMUS.F90
file geometry.surveys.Muzzin-2013-ULTRAVISTA.F90
file geometry.surveys.full_sky.F90
file hot_halo.outflow_-reincorporation.Henriques2013.F90
file intergalactic_medium.filtering_-mass.Gnedin2000.F90

module intergalactic_medium_state
file merger_trees.construct.build.F90
file merger_trees.construct.read.F90
function importerunitconvert1d
function importerunitconvertscalar
file merger_trees.construct.read.importer.galacticus.F90
file merger_trees.evolve.timesteps.history.F90
file merger_trees.evolve.timesteps.simple.F90
file merger_trees.evolver.standard.F90
function augmentconstructorinternal
file merger_trees.operators.export.F90
function outputrootmassesconstructorparameters
file merger_trees.operators.profiler.F90
file merger_trees.operators.regrid_times.F90
subroutine merger_trees_render_dump
function sedfitevaluate
file models.likelihoods.mass_function.F90
file models.likelihoods.spin_distribution.F90
file nodes.property_extractor.ICM_SZ.F90
file nodes.property_extractor.density_contrasts.F90
file nodes.property_extractor.redshift.F90
module node_component_disk_very_simple
file merger_trees.branching_probability.generalized_Press_Schechter.F90
file merger_trees.construct.builder.Cole2000.F90
subroutine readphasespacepositionrealize
function importerunitconvert2d
file merger_trees.construct.read.importer.SussingMergerTrees.F90
file merger_trees.construct.smooth_accretion.F90
file merger_trees.evolve.timesteps.record_evolution.F90
file merger_trees.evolve.timesteps.standard.F90
file merger_trees.operators.augment.F90
file merger_trees.operators.conditional_mass_function.F90
file merger_trees.operators.mass_accretion_history.F90
file merger_trees.operators.particulate.F90
function prunebytimeconstructorparameters
file merger_trees.outputter.standard.F90
file models.likelihoods.SED_fit.F90
file models.likelihoods.halo_mass_function.F90
file models.likelihoods.projected_correlation_function.F90
function spindistributionconstructorinternal
file nodes.property_extractor.ICM_Xray_luminosity.F90
file nodes.property_extractor.lightcone.F90
module node_component_basic_standard_extended
module node_component_hot_halo_standard

subroutine stellar_luminosities_-initialize
file output.times.list.F90
file radiation.intergalactic_-background.file.F90
file satellites.merging.timescale.Villalobos_-2013.F90
file satellites.merging.timescale.Wetzel-White.F90
file satellites.merging.virial_-orbits.Jiang2014.F90
file star_-formation.feedback.disks.halo_-scaling.F90
file star_-formation.feedback.disks.velocity_-maximum_scaling.F90
file star_-formation.feedback.spheroids.velocity_-maximum_scaling.F90
file star_-formation.timescales.disks.halo_-scaling.F90
file star_-formation.timescales.spheroids.velocity_-maximum_scaling.F90
module cosmological_density_field
file structure_formation.critical_-overdensity.fixed.F90
function sphericalcollapsematterlambdagradientmass
file structure_-formation.gravitational_-lensing.Takahashi_2011.F90
file structure_formation.halo_-environment.normal.F90
file structure_formation.halo_mass_-function.Tinker2008Form.F90
file structure_formation.power_-spectrum.nonlinear.CosmicEmu.F90
subroutine stellar_luminosities_-special_cases
file radiation.cosmic_microwave_-background.F90
file radiation.intergalactic_-background.internal.F90
function villalobos2013timeuntilmerging
file satellites.merging.virial_-orbits.Benson2005.F90
file satellites.merging.virial_-orbits.Wetzel2010.F90
file star_-formation.feedback.disks.power_-law.redshift_scaling.F90
file star_-formation.feedback.spheroids.power_-law.redshift_scaling.F90
file star_-formation.timescales.disks.Baugh2005.F90
file star_-formation.timescales.disks.velocity_-maximum_scaling.F90
file statistics.Nbody.halos.mass_-errors.Trenti2010.F90
file structure_formation.critical_-overdensity.Kitayama_Suto1996.F90
file structure_formation.critical_-overdensity.spherical_collapse_-matter_lambda.F90
file structure_formation.excursion_-sets.first_crossing_-distribution.Farahi.F90
file structure_formation.halo_-environment.lognormal.F90
file structure_formation.halo_mass_-function.Despali_2015.F90
file structure_formation.linear_-growth.simple.F90
file structure_formation.power_-spectrum.nonlinear.PeacockDodds1996.F90

function peacockdodds1996value

subroutine make_table

subroutine spherical_collapse_matter_-lambda_critical_overdensity_tabulate
function bbksconstructorparameters
function eisensteinhu1999constructorparameters
function identityconstructorparameters

function bryannorman1998densitycontrast

file structure_formation.virial_-density_contrast.Kitayama_-Suto1996.F90
file structure_formation.virial_-density_contrast.percolation.F90
module virial_density_contrast_-percolation_utilities

file tasks.catalog_projected_-correlation_function.F90
file tasks.excursion_sets.F90
file tasks.halo_model.projected_-correlation_function.F90
file tasks.halo_spin_distribution.F90

file tasks.mass_function_covariance.F90
program test_diemerkravtsov2014_-concentration
program test_prada2011_concentration
program test_zhao2009_dark_energy
program tests_comoving_distance
program test_cooling_functions
program test_dark_matter_profiles

program tests_halo_mass_function_-tinker
program tests_linear_growth_-cosmological_constant
program tests_linear_growth_open
program tests_power_spectrum

module spherical_collapse_matter_-dark_energy
subroutine spherical_collape_matter_-lambda_delta_virial_tabulate
subroutine spherical_collapse_matter_-lambda_nonlinear_mapping
function bode2001constructorparameters
file structure_formation.transfer_-function.file.F90
file structure_formation.virial_-density_contrast.Bryan_Norman.F90
function bryannorman1998densitycontrastrateofchange
file structure_formation.virial_-density_contrast.fixed.F90

function percolationconstructorparameters
file structure_formation.virial_-density_contrast.spherical_collapse_-matter_lambda.F90
file tasks.conditional_mass_-function.F90
file tasks.halo_mass_function.F90
file tasks.halo_model_generate.F90

file tasks.intergalactic_medium_-state.F90
file tasks.power_spectrum.F90
program test_nfw96_concentration_-dark_energy
program test_zhao2009_flat
program test_zhao2009_open
program test_correa2015_concentration
program tests_cosmic_age
program test_dark_matter_profiles_-generic
program tests_linear_growth_eds

program tests_linear_growth_dark_-energy
program test_correa2015_mah
program tests_spherical_collapse_-dark_energy_eds

program tests_spherical_collapse_dark_energy_omega_zero_point_six
program tests_spherical_collapse_dark_energy_omega_zero_point_eight
program tests_spherical_collapse_dark_energy_omega_half
program tests_spherical_collapse_dark_energy_omega_two_thirds
program tests_spherical_collapse_dark_energy_lambda
program tests_spherical_collapse_dark_energy_open
program tests_spherical_collapse_flat
program tests_spherical_collapse_nonlinear
program tests_spherical_collapse_open
program test_stellar_populations_luminosities
program tests_transfer_functions
file universe.operators.intergalactic_medium_state_evolve.F90
module io_irate

file: cosmology.functions.matter_dark_energy.F90

Description: An implementation of the cosmological functions class for cosmologies consisting of collisionless matter and dark energy with an equation of state of the form: $P = \rho^w$ with $w(a) = w_0 + w_1 a(1-a)$.
Code lines: 682
Modules used: cosmology_parameters fgsl

interface: cosmologyfunctionsmatterdarkenergy

Description: Constructors for the matter plus dark energy cosmological functions class.
Code lines: 4
Contained by: file cosmology.functions.matter_dark_energy.F90

function: matterdarkenergyagetableodes

Description: System of differential equations to solve for expansion factor vs. age.
Code lines: 13
Contained by: file cosmology.functions.matter_dark_energy.F90

function: matterdarkenergyconstructorinternal

Description: Constructor for the matter plus dark energy cosmological functions class.
Code lines: 15
Contained by: file cosmology.functions.matter_dark_energy.F90
Modules used: cosmology_parameters

function: matterdarkenergyconstructorparameters

Description: Default constructor for the matter plus dark energy cosmological functions class.
Code lines: 31
Contained by: file cosmology.functions.matter_dark_energy.F90
Modules used: input_parameters

function: matterdarkenergycosmictime

Description: Return the cosmological matter density in units of the critical density at the present day.
Code lines: 36
Contained by: file cosmology.functions.matter_dark_energy.F90
Modules used: galacticus_error numerical_interpolation

function: `matterdarkenergydistancecomoving`
Description: Returns the comoving distance to cosmological time `time`.
Code lines: 11
Contained by: file `cosmology.functions.matter_dark_energy.F90`
Modules used: `galacticus_error`

function: `matterdarkenergydistancecomovingconvert`
Description: Convert bewteen different measures of distance.
Code lines: 12
Contained by: file `cosmology.functions.matter_dark_energy.F90`
Modules used: `galacticus_error`

function: `matterdarkenergydomination`
Description: Function used in root finding when seeking epoch at which matter dominates over dark energy.
Code lines: 7
Contained by: file `cosmology.functions.matter_dark_energy.F90`

function: `matterdarkenergydominationepochmatter`
Code lines: 59
Contained by: file `cosmology.functions.matter_dark_energy.F90`
Modules used: `cosmology_functions_parameters` `root_finder`

function: `matterdarkenergyequalityepochmatterdarkenergy`
Description: Return the epoch of matter-dark energy magnitude equality (either expansion factor or cosmic time).
Code lines: 54
Contained by: file `cosmology.functions.matter_dark_energy.F90`
Modules used: `cosmology_functions_parameters` `cosmology_parameters` `root_finder`

function: `matterdarkenergyequationofstatedarkenergy`
Description: Return the dark energy equation of state.
Code lines: 21
Contained by: file `cosmology.functions.matter_dark_energy.F90`
Modules used: `galacticus_error`

function: `matterdarkenergyexpansionfactorchange`
Description: Compute the expansion factor at time `timeEnd` given an initial value `expansionFactorStart` at time `timeStart`.
Code lines: 21
Contained by: file `cosmology.functions.matter_dark_energy.F90`
Modules used: `ode_solver`

function: `matterdarkenergyexponentdarkenergy`
Description: Return the dark energy exponent.
Code lines: 21
Contained by: file `cosmology.functions.matter_dark_energy.F90`

Modules used: galacticus_error

function: matterdarkenergyexponentdarkenergyderivative
Description: Return the derivative of the dark energy exponent with respect to expansion factor.
Code lines: 22
Contained by: file cosmology.functions.matter_dark_energy.F90
Modules used: galacticus_error

function: matterdarkenergyhubbleparameterepochal
Description: Returns the Hubble parameter at the request cosmological time, time, or expansion factor, expansionFactor.
Code lines: 37
Contained by: file cosmology.functions.matter_dark_energy.F90
Modules used: galacticus_error

function: matterdarkenergyhubbleparameterrateofchange
Description: Returns the rate of change of the Hubble parameter at the requested cosmological time, time, or expansion factor, expansionFactor.
Code lines: 26
Contained by: file cosmology.functions.matter_dark_energy.F90
Modules used: galacticus_error

subroutine: matterdarkenergymakeexpansionfactortable
Description: Builds a table of expansion factor vs. time for dark energy universes.
Code lines: 127
Contained by: file cosmology.functions.matter_dark_energy.F90
Modules used: cosmology_parameters memory_management numerical_interpolation numerical_ranges

function: matterdarkenergyomegadarkenergyepochal
Description: Return the dark energy density parameter at expansion factor expansionFactor.
Code lines: 26
Contained by: file cosmology.functions.matter_dark_energy.F90
Modules used: galacticus_error

subroutine: matterdarkenergytargetself
Description: Set a module-scope pointer to the current dark energy cosmology functions object.
Code lines: 7
Contained by: file cosmology.functions.matter_dark_energy.F90

function: matterdarkenergytimeatdistancecomoving
Description: Returns the cosmological time corresponding to given comovingDistance.
Code lines: 11
Contained by: file cosmology.functions.matter_dark_energy.F90
Modules used: galacticus_error

file: cosmology.functions.matter_lambda.F90
Description: An implementation of the cosmological functions class for cosmologies consisting of collisionless matter plus a cosmological constant.
Code lines: 1061

Modules used: cosmology_parameters fgsl

interface: cosmologyfunctionsmatterlambda
Description: Constructors for the matter plus cosmological constant cosmological functions class.
Code lines: 4
Contained by: file cosmology.functions.matter_lambda.F90

function: matterlambdaagetableodes
Description: System of differential equations to solve for expansion factor vs. age.
Code lines: 11
Contained by: file cosmology.functions.matter_lambda.F90

function: matterlambdacollapseodes
Description: System of differential equations to solve for age vs. expansion factor.
Code lines: 14
Contained by: file cosmology.functions.matter_lambda.F90

function: matterlambdacomovingdistanceintegrand
Description: Integrand function used in computing the comoving distance.
Code lines: 9
Contained by: file cosmology.functions.matter_lambda.F90
Modules used: numerical_constants_astronomical numerical_constants_physical

function: matterlambdaconstructorinternal
Description: Constructor for the matter plus cosmological constant cosmological functions class.
Code lines: 96
Contained by: file cosmology.functions.matter_lambda.F90
Modules used: iso_varying_string numerical_comparison ode_solver

function: matterlambdaconstructorparameters
Description: Parameter-based constructor for the matter plus cosmological constant cosmological functions class.
Code lines: 13
Contained by: file cosmology.functions.matter_lambda.F90
Modules used: input_parameters

function: matterlambdacosmictime
Description: Return the cosmological matter density in units of the critical density at the present day.
Code lines: 41
Contained by: file cosmology.functions.matter_lambda.F90
Modules used: galacticus_error numerical_interpolation

subroutine: matterlambdadensityscalingearlytime
Code lines: 14
Contained by: file cosmology.functions.matter_lambda.F90

subroutine: matterlambdadestructor
Description: Default constructor for the matter plus cosmological constant cosmological functions class.

Code lines: 20
Contained by: file cosmology.functions.matter_lambda.F90
Modules used: numerical_interpolation

function: matterlambdadistanceangular
Description: Returns the angular diameter distance to cosmological time time.
Code lines: 11
Contained by: file cosmology.functions.matter_lambda.F90
Modules used: galacticus_error numerical_interpolation

function: matterlambdadistancecomoving
Description: Returns the comoving distance to cosmological time time.
Code lines: 34
Contained by: file cosmology.functions.matter_lambda.F90
Modules used: galacticus_error numerical_interpolation

function: matterlambdadistancecomovingconvert
Description: Convert between different measures of distance.
Code lines: 54
Contained by: file cosmology.functions.matter_lambda.F90
Modules used: cosmology_functions_options galacticus_error numerical_interpolation

function: matterlambdadistanceluminosity
Description: Returns the luminosity distance to cosmological time time.
Code lines: 11
Contained by: file cosmology.functions.matter_lambda.F90
Modules used: galacticus_error numerical_interpolation

function: matterlambdadominationepochmatter
Code lines: 28
Contained by: file cosmology.functions.matter_lambda.F90
Modules used: cosmology_functions_parameters

subroutine: matterlambdaepochvalidate
Description: Validate a cosmic epoch, specified either by time or expansion factor, and optionally return time, expansion factor, and collapsing status.
Code lines: 42
Contained by: file cosmology.functions.matter_lambda.F90
Modules used: galacticus_error

function: matterlambdaequalityepochmattercurvature
Description: Return the epoch of matter-curvature magnitude equality (either expansion factor or cosmic time).
Code lines: 16
Contained by: file cosmology.functions.matter_lambda.F90
Modules used: cosmology_functions_parameters

function: matterlambdaequalityepochmatterdarkenergy

Description: Return the epoch of matter-dark energy magnitude equality (either expansion factor or cosmic time).
Code lines: 16
Contained by: file cosmology.functions.matter_lambda.F90
Modules used: cosmology_functions_parameters

function: matterlambdaequalityepochmatterradiation
Description: Return the epoch of matter-radiation magnitude equality (either expansion factor or cosmic time).
Code lines: 16
Contained by: file cosmology.functions.matter_lambda.F90
Modules used: cosmology_functions_parameters

function: matterlambdaequationofstatedarkenergy
Description: Return the dark energy equation of state.
Code lines: 9
Contained by: file cosmology.functions.matter_lambda.F90

function: matterlambdaexpansionfactor
Description: Returns the expansion factor at cosmological time time.
Code lines: 70
Contained by: file cosmology.functions.matter_lambda.F90
Modules used: galacticus_error iso_varying_string numerical_interpolation

function: matterlambdaexpansionrate
Description: Returns the cosmological expansion rate, $\dot{a}/a$ at expansion factor expansionFactor.
Code lines: 9
Contained by: file cosmology.functions.matter_lambda.F90

function: matterlambdaexponentdarkenergy
Description: Return the dark energy equation of state.
Code lines: 9
Contained by: file cosmology.functions.matter_lambda.F90

function: matterlambdahubbleparameterepochal
Description: Returns the Hubble parameter at the request cosmological time, time, or expansion factor, expansionFactor.
Code lines: 33
Contained by: file cosmology.functions.matter_lambda.F90
Modules used: galacticus_error

function: matterlambdahubbleparameterrateofchange
Description: Returns the rate of change of the Hubble parameter at the request cosmological time, time, or expansion factor, expansionFactor.
Code lines: 13
Contained by: file cosmology.functions.matter_lambda.F90
Modules used: galacticus_error

subroutine: matterlambdamakedistancetable

Description: Builds a table of distance vs. time.
Code lines: 59
Contained by: file cosmology.functions.matter_lambda.F90
Modules used: memory_management numerical_integration numerical_interpolation numerical_ranges

subroutine: matterlambdamakeexpansionfactortable
Description: Builds a table of expansion factor vs. time.
Code lines: 106
Contained by: file cosmology.functions.matter_lambda.F90
Modules used: memory_management numerical_interpolation numerical_ranges ode_solver

function: matterlambdamatterdensityepochal
Description: Return the matter density at expansion factor expansionFactor.
Code lines: 28
Contained by: file cosmology.functions.matter_lambda.F90
Modules used: galacticus_error

function: matterlambdaomegadarkenergyepochal
Description: Return the dark energy density parameter at expansion factor expansionFactor.
Code lines: 13
Contained by: file cosmology.functions.matter_lambda.F90
Modules used: galacticus_error

function: matterlambdaomegamatterepochal
Description: Return the matter density parameter at expansion factor expansionFactor.
Code lines: 13
Contained by: file cosmology.functions.matter_lambda.F90
Modules used: galacticus_error

function: matterlambdaomegamatterrateofchange
Description: Return the rate of change of the matter density parameter at expansion factor expansionFactor.
Code lines: 13
Contained by: file cosmology.functions.matter_lambda.F90
Modules used: galacticus_error

function: matterlambdatemperaturecmbepochal
Description: Return the temperature of the CMB at expansion factor expansionFactor.
Code lines: 13
Contained by: file cosmology.functions.matter_lambda.F90
Modules used: galacticus_error

function: matterlambdatimeatdistancecomoving
Description: Returns the cosmological time corresponding to given comovingDistance.
Code lines: 32
Contained by: file cosmology.functions.matter_lambda.F90
Modules used: galacticus_error numerical_interpolation

function: `matterlambdatimebigcrunch`
Description: Return the time of the Big Crunch (or a negative value if no Big Crunch occurs).
Code lines: 11
Contained by: file cosmology.functions.matter_lambda.F90

file: `cosmology.functions.options.F90`
Description: Contains a module which provides options for cosmological functions.
Code lines: 29

module: `cosmology_functions_options`
Description: Provides options for cosmological functions.
Code lines: 7
Contained by: file cosmology.functions.options.F90
Used by:

function matterlambdadistancecomovingconvert	function staticuniversedistancecomovingconvert
function baldry2012gamaconstructorinternal	function caputi2011ukidssudsconstructorinternal
function caputi2011ukidssudsdistancemaximum	function davidzon2013vipersconstructorinternal
function gunawardhana2013sdssdistancemaximum	function hearin2014sdssconstructorinternal
function kelvin2014gamanearconstructorinternal	function liwhite2009sdssconstructorinternal
function liwhite2009sdssdistancemaximum	function monterodorta2009sdssdistancemaximum
function monterodorta2009sdssdistanceminimum	function moustakas2013primusconstructorinternal
function moustakas2013primusdistancemaximum	function muzzin2013ultravistaconstructorinternal
function muzzin2013ultravistadistancemaximum	function tomczak2014zfourgeconstructorinternal
function tomczak2014zfourgedistancemaximum	function fullskyconstructorinternal
program tests_comoving_distance	

file: `cosmology.functions.parameters.F90`
Description: Contains a module which option parameters for cosmology functions.
Code lines: 31

module: `cosmology_functions_parameters`
Description: Defines option parameters for cosmology functions.
Code lines: 9
Contained by: file cosmology.functions.parameters.F90
Used by:

function matterdarkenergydominationepochmatter	function matterdarkenergyequalityepochmatterdarkenergy
function matterlambdadominationepochmatter	function matterlambdaequalityepochmattercurvature

function matterlambdaequalityepochmatterdarkenergy function matterlambdaequalityepochmatterradiation function bode2001constructorparameters

file: `cosmology.functions.static_universe.F90`
Description: An implementation of the cosmological functions class for static universes. Intended for testing purposes.
Code lines: 456
Modules used: cosmology_parameters

interface: `cosmologyfunctionsstaticuniverse`
Description: Constructors for the matter plus cosmological constant cosmological functions class.
Code lines: 4
Contained by: file cosmology.functions.static_universe.F90

function: `staticuniverseconstructorinternal`
Description: Constructor for the matter plus cosmological constant cosmological functions class.
Code lines: 8
Contained by: file cosmology.functions.static_universe.F90

function: `staticuniverseconstructorparameters`
Description: Parameter-based constructor for the matter plus cosmological constant cosmological functions class.
Code lines: 13
Contained by: file cosmology.functions.static_universe.F90
Modules used: input_parameters

function: `staticuniversecosmictime`
Description: Return the cosmological time at a given expansion factor.
Code lines: 12
Contained by: file cosmology.functions.static_universe.F90
Modules used: galacticus_error

subroutine: `staticuniversedensityscalingearlytime`
Code lines: 14
Contained by: file cosmology.functions.static_universe.F90
Modules used: galacticus_error

subroutine: `staticuniversedestructor`
Description: Default constructor for the matter plus cosmological constant cosmological functions class.
Code lines: 7
Contained by: file cosmology.functions.static_universe.F90

function: `staticuniversedistanceangular`
Description: Returns the angular diameter distance to cosmological time `time`.
Code lines: 11
Contained by: file cosmology.functions.static_universe.F90
Modules used: galacticus_error numerical_interpolation

function: `staticuniversedistancecomoving`
Description: Returns the comoving distance to cosmological time `time`.
Code lines: 11
Contained by: file `cosmology.functions.static_universe.F90`
Modules used: `galacticus_error`

function: `staticuniversedistancecomovingconvert`
Description: Convert bewteen different measures of distance.
Code lines: 36
Contained by: file `cosmology.functions.static_universe.F90`
Modules used: `cosmology_functions_options` `galacticus_error`

function: `staticuniversedistanceluminosity`
Description: Returns the luminosity distance to cosmological time `time`.
Code lines: 11
Contained by: file `cosmology.functions.static_universe.F90`
Modules used: `galacticus_error` `numerical_interpolation`

function: `staticuniversedominationepochmatter`
Code lines: 10
Contained by: file `cosmology.functions.static_universe.F90`
Modules used: `galacticus_error`

subroutine: `staticuniverseepochvalidate`
Description: Validate a cosmic epoch, specified either by time or expansion factor, and optionally return time, expansion factor, and collapsing status.
Code lines: 35
Contained by: file `cosmology.functions.static_universe.F90`
Modules used: `galacticus_error`

function: `staticuniverseequalityepochmattercurvature`
Description: Return the epoch of matter-curvature magnitude equality (either expansion factor or cosmic time).
Code lines: 11
Contained by: file `cosmology.functions.static_universe.F90`
Modules used: `galacticus_error`

function: `staticuniverseequalityepochmatterdarkenergy`
Description: Return the epoch of matter-dark energy magnitude equality (either expansion factor or cosmic time).
Code lines: 11
Contained by: file `cosmology.functions.static_universe.F90`
Modules used: `galacticus_error`

function: `staticuniverseequalityepochmatterradiation`
Description: Return the epoch of matter-radiation magnitude equality (either expansion factor or cosmic time).
Code lines: 11
Contained by: file `cosmology.functions.static_universe.F90`

Modules used: galacticus_error

function: staticuniverseequationofstatedarkenergy
Description: Return the dark energy equation of state.
Code lines: 9
Contained by: file cosmology.functions.static_universe.F90

function: staticuniverseexpansionfactor
Description: Returns the expansion factor at cosmological time time.
Code lines: 9
Contained by: file cosmology.functions.static_universe.F90

function: staticuniverseexpansionrate
Description: Returns the cosmological expansion rate, $\dot{a}/a$ at expansion factor expansionFactor.
Code lines: 9
Contained by: file cosmology.functions.static_universe.F90

function: staticuniverseexponentdarkenergy
Description: Return the dark energy equation of state.
Code lines: 9
Contained by: file cosmology.functions.static_universe.F90

function: staticuniversehubbleparameterepochal
Description: Returns the Hubble parameter at the request cosmological time, time, or expansion factor, expansionFactor.
Code lines: 10
Contained by: file cosmology.functions.static_universe.F90

function: staticuniversehubbleparameterrateofchange
Description: Returns the rate of change of the Hubble parameter at the request cosmological time, time, or expansion factor, expansionFactor.
Code lines: 10
Contained by: file cosmology.functions.static_universe.F90

function: staticuniversematterdensityepochal
Description: Return the matter density at expansion factor expansionFactor.
Code lines: 10
Contained by: file cosmology.functions.static_universe.F90

function: staticuniverseomegadarkenergyepochal
Description: Return the dark energy density parameter at expansion factor expansionFactor.
Code lines: 12
Contained by: file cosmology.functions.static_universe.F90
Modules used: galacticus_error

function: staticuniverseomegamatterepochal
Description: Return the matter density parameter at expansion factor expansionFactor.
Code lines: 12
Contained by: file cosmology.functions.static_universe.F90

Modules used: galacticus_error

function: staticuniverseomegamatterrateofchange
Description: Return the rate of change of the matter density parameter at expansion factor expansionFactor.
Code lines: 12
Contained by: file cosmology.functions.static_universe.F90
Modules used: galacticus_error

function: staticuniversetemperaturecmbepochal
Description: Return the temperature of the CMB at expansion factor expansionFactor.
Code lines: 11
Contained by: file cosmology.functions.static_universe.F90
Modules used: galacticus_error

function: staticuniversetimeatdistancecomoving
Description: Returns the cosmological time corresponding to given comovingDistance.
Code lines: 11
Contained by: file cosmology.functions.static_universe.F90
Modules used: galacticus_error

function: staticuniversetimebigcrunch
Description: Return the time of the Big Crunch in a static cosmology. Since no Big Crunch occurs we return a negative value to indicate that.
Code lines: 9
Contained by: file cosmology.functions.static_universe.F90

file: cosmology.parameters.F90
Description: Contains a module which provides an object that implements cosmological parameters.
Code lines: 84

module: cosmology_parameters
Description: Provides an object that implements cosmological parameters.
Code lines: 62
Contained by: file cosmology.parameters.F90
Used by: file accretion.halo.simple.F90; program benchmark_stellar_populations_luminosities; file cosmology.functions.matter_dark_energy.F90; function matterdarkenergyconstructorinternal; function matterdarkenergyequalityepochmatterdarkenergy; subroutine matterdarkenergymakeexpansionfactortable; file cosmology.functions.matter_lambda.F90; file cosmology.functions.static_universe.F90; file dark_matter.particle.cold_dark_matter.F90; file dark_matter.particle.warm_dark_matter.thermal.F90; file dark_matter_halos.scales.virial_density_contrast.F90; file dark_matter_profiles.adiabatic_Gnedin2004.F90

file dark_matter_profiles.dark_matter_only.F90
file dark_matter_profiles.structure.concentration.Correa2015.F90
file dark_matter_profiles.structure.concentration.Dutton-Macciò2014.F90
file dark_matter_profiles.structure.concentration.Klypin2015.F90
file dark_matter_profiles.structure.concentration.MunozCuartas2011.F90
file dark_matter_profiles.structure.concentration.Prada2011.F90
function dark_matter_profile_mass_definition
file dark_matter_profiles.structure.scale.concentration.F90
function hivshalomassrelationpadmanabhan2017constructorinternal
function blackholebulgerelationconstructorinternal
function concentrationdistributioncdmcococonstructorinternal
function concentrationvshalomasscdmludlow2016constructorinternal
function correlationfunctionconstructorfile
function galaxysizessdssconstructorinternal
function luminosityfunctionsobral2013hizelsconstructorinternal
function massfunctionhialfalfamartin2010constructorinternal
function massfunctionstellarbernardi2013sdssconstructorinternal
function massfunctionstellarprimusconstructorinternal
function massfunctionstellarukidssudsconstructorinternal
function massfunctionstellarvipersconstructorinternal
function massmetallicityandrews2013constructorinternal
function morphologicalfractiongamamoffett2016constructorinternal
file dark_matter_profiles.structure.concentration.Bullock2001.F90
file dark_matter_profiles.structure.concentration.Diemer-Kravtsov2014.F90
file dark_matter_profiles.structure.concentration.Gao2008.F90
file dark_matter_profiles.structure.concentration.Ludlow2016_fit.F90
file dark_matter_profiles.structure.concentration.NFW.F90
file dark_matter_profiles.structure.concentration.Zhao2009.F90
file dark_matter_profiles.structure.scale.Ludlow2016.F90
subroutine galacticus_function_classes_destroy
function hivshalomassrelationpadmanabhan2017constructorparameters
function colordistributionsdssconstructorinternal
function concentrationdistributioncdmcococonstructorparameters
function concentrationvshalomasscdmludlow2016constructorparameters
function correlationfunctionhearin2013sdssconstructorinternal
function luminosityfunctiongunawardhana2013sdssconstructorinternal
function luminosityfunctionmonterodorta2009sdssconstructorinternal
function massfunctionhialfalfamartin2010constructorparameters
function massfunctionstellarbaldry2012gamaconstructorinternal
function massfunctionstellarsdssconstructorinternal
function massfunctionstellarultravistaconstructorinternal
function massfunctionstellarzfourgeconstructorinternal
function massmetallicityblanc2017constructorinternal
function starformationmainsequencerelationleauthaud2012constructorinternal

function stellarvshalomassrelationleauthaud2012constructorparameters
subroutine galacticus_state_store
file geometry.surveys.Martin-2010-ALFALFA.F90
subroutine martin2010alfalfarandomsinitialize
file hot_halo.mass_-distribution.cored.core_-radius.growing.F90
file intergalactic_medium.filtering_-mass.Gnedin2000.F90
module intergalactic_medium_state
file merger_trees.construct.build.F90
function importerunitconvert1d
function importerunitconvertscalar
file merger_-trees.construct.read.importer.galacticus.F90
file merger_-trees.operators.particulate.F90
file nodes.property_extractor.density_-contrasts.F90
module node_component_black_hole_-standard
module node_component_hot_halo_-standard
subroutine node_component_hot_halo_vs_-delayed_tree_initialize
file satellites.merging.virial_-orbits.Jiang2014.F90
file structure_formation.cosmological_-mass_variance.filtered_power_-spectrum.F90
file structure_formation.critical_-overdensity.warm_dark_matter.F90
file structure_formation.halo_-environment.lognormal.F90
module halo_mass_functions
file structure_formation.power_-spectrum.nonlinear.CosmicEmu.F90
subroutine galacticus_state_retrieve
function squareconstructorparameters
function martin2010alfalfadistancemaximum
file halo_model.power_spectrum_-modifier.triaxiality.F90
subroutine interface_camb_transfer_-function
subroutine gnedin2000conditionsinitialodes
function recfastconstructorinternal
file merger_-trees.construct.build.masses.fixed_-mass.F90
function importerunitconvert2d
file merger_-trees.construct.read.importer.SussingMergerTrees.F90
file merger_trees.operators.export.F90
file models.likelihoods.halo_mass_-function.F90
module node_component_basic_standard_-extended
module node_component_hot_halo_cold_-mode
subroutine node_component_hot_halo_-very_simple_tree_initialize
file radiation.intergalactic_-background.internal.F90
file satellites.tidal_-heating.rate.Gnedin1999.F90
file structure_formation.cosmological_-mass_variance.peak_background_-split.F90
file structure_-formation.gravitational_-lensing.Takahashi_2011.F90
file structure_formation.halo_-environment.normal.F90
file structure_formation.linear_-growth.simple.F90
file structure_formation.power_-spectrum.variance.window_-function.Lagrangian_Chan2017.F90

function lagrangianchan2017value
file structure_formation.power_spectrum.variance.window_function.sharp_kSpace.F90
file structure_formation.power_spectrum.variance.window_function.smooth_k_space.F90
function smoothkspacevalue
file structure_formation.power_spectrum.variance.window_function.top_hat.F90
function tophatvalue
file structure_formation.power_spectrum.variance.window_function.top_hat_kspace_sharp_hybrid.F90
file structure_formation.transfer_function.BBKS.F90
file structure_formation.transfer_function.BBKS.WDM.F90
file structure_formation.transfer_function.Bode2001.F90
file structure_formation.transfer_function.CAMB.F90
file structure_formation.transfer_function.Eisenstein_Hu.F90
file structure_formation.transfer_function.file.F90
file structure_formation.virial_density_contrast.Bryan_Norman.F90
file structure_formation.virial_density_contrast.fixed.F90
module virial_density_contrast_percolation_utilities
file tasks.catalog_projected_correlation_function.F90
file tasks.excursion_sets.F90
file tasks.halo_mass_function.F90
file tasks.halo_model_generate.F90
file tasks.merger_tree_file_builder.F90
file tasks.power_spectrum.F90
program test_diemerkravtsov2014_concentration
program test_nfw96_concentration_dark_energy
program test_prada2011_concentration
program tests_comoving_distance
program tests_cosmic_age
program test_dark_matter_profiles_generic
program tests_halo_mass_function_tinker
program test_parameters
program tests_power_spectrum
program tests_sigma
program tests_spherical_collapse_nonlinear
program test_stellar_populations_luminosities
program tests_transfer_functions
file universe.operators.intergalactic_medium_state_evolve.F90
module io_irate
subroutine iratereadhalos

file: cosmology.parameters.simple.F90
Description: A simple implementation of the cosmological parameters class.
Code lines: 218

interface: cosmologyparameterssimple
Description: Constructors for the simple cosmological parameters class.
Code lines: 4
Contained by: file cosmology.parameters.simple.F90

function: `simpleconstructorinternal`
Description: Internal constructor for the simple cosmological parameters class.
Code lines: 12
Contained by: file `cosmology.parameters.simple.F90`

function: `simpleconstructorparameters`
Description: Constructor for the simple cosmological parameters class which takes a parameter set as input.
Code lines: 61
Contained by: file `cosmology.parameters.simple.F90`
Modules used: `galacticus_error`

function: `simpledensitycritical`
Description: Return the present day critical density of the Universe in units of $M_\odot/\mathrm{Mpc}^3$.
Code lines: 9
Contained by: file `cosmology.parameters.simple.F90`
Modules used: `numerical_constants_math` `numerical_constants_physical`

function: `simplehubbleconstant`
Description: Return the present day value of the Hubble constant.
Code lines: 23
Contained by: file `cosmology.parameters.simple.F90`
Modules used: `galacticus_error` `numerical_constants_astronomical` `numerical_constants_prefixes`

function: `simpleomegabaryon`
Description: Return the cosmological baryon density in units of the critical density at the present day.
Code lines: 8
Contained by: file `cosmology.parameters.simple.F90`
Modules used: `galacticus_error`

function: `simpleomegacurvature`
Description: Return the cosmological curvature density in units of the critical density at the present day.
Code lines: 7
Contained by: file `cosmology.parameters.simple.F90`

function: `simpleomegadarkenergy`
Description: Return the cosmological dark energy density in units of the critical density at the present day.
Code lines: 8
Contained by: file `cosmology.parameters.simple.F90`
Modules used: `galacticus_error`

function: `simpleomegamatter`
Description: Return the cosmological matter density in units of the critical density at the present day.
Code lines: 8
Contained by: file `cosmology.parameters.simple.F90`
Modules used: `galacticus_error`

function: `simpleomegaradiation`
Description: Return the cosmological radiation density in units of the critical density at the present day.
Code lines: 9
Contained by: file cosmology.parameters.simple.F90
Modules used: numerical_constants_astronomical numerical_constants_physical

function: `simpletemperaturecmb`
Description: Return the present day temperature of the CMB.
Code lines: 7
Contained by: file cosmology.parameters.simple.F90

file: `dark_matter.particle.F90`
Description: Contains a module which provides a class that implements dark matter particle physics.
Code lines: 33

module: `dark_matter_particles`
Description: Provides a class that implements dark matter parrticle physics.
Code lines: 11
Contained by: file dark_matter.particle.F90
Used by: subroutine galacticus_function_classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; file structure_formation.critical_overdensity.Kitayama_Suto1996.F90; function kitayamasuto1996constructorinternal; function sphericalcollapsematterdeconstructorinternal; function sphericalcollapsematterdeconstructorparameters; file structure_formation.critical_overdensity.spherical_collapse_matter_lambda.F90; function sphericalcollapsematterlambdaconstructorinternal; file structure_formation.critical_overdensity.warm_dark_matter.F90; file structure_formation.transfer_function.BBKS.F90; file structure_formation.transfer_function.BBKS.WDM.F90; file structure_formation.transfer_function.Bode2001.F90; file structure_formation.transfer_function.CAMB.F90; function cambconstructorinternal; file structure_formation.transfer_function.Eisenstein_Hu.F90; function eisensteinhu1999constructorinternal; program tests_power_spectrum; program tests_transfer_functions

file: `dark_matter.particle.cold_dark_matter.F90`
Description: Contains a module which implements a cold dark matter particle class.
Code lines: 50
Modules used: cosmology_parameters

function: `cdmconstructorparameters`
Description: Constructor for the "`CDM`" dark matter particle class which takes a parameter set as input.

Code lines: 10
Contained by: file dark_matter.particle.cold_dark_matter.F90
Modules used: input_parameters

interface: darkmatterparticlecdm
Description: Constructors for the "CDM" dark matter particle class.
Code lines: 3
Contained by: file dark_matter.particle.cold_dark_matter.F90

file: dark_matter.particle.warm_dark_matter.thermal.F90
Description: Contains a module which implements a thermal warm dark matter particle class.
Code lines: 154
Modules used: cosmology_parameters

interface: darkmatterparticlewdmthermal
Description: Constructors for the "WDMThermal" dark matter particle class.
Code lines: 4
Contained by: file dark_matter.particle.warm_dark_matter.thermal.F90

function: wdmthermalconstructorinternal
Description: Internal constructor for the "WDMThermal" dark matter particle class.
Code lines: 12
Contained by: file dark_matter.particle.warm_dark_matter.thermal.F90
Modules used: input_parameters

function: wdmthermalconstructorparameters
Description: Constructor for the "WDMThermal" dark matter particle class which takes a parameter set as input.
Code lines: 32
Contained by: file dark_matter.particle.warm_dark_matter.thermal.F90
Modules used: input_parameters

function: wdmthermaldegreesoffreedomeffective
Description: Return the effective number of degrees of freedom of a thermal warm dark matter particle.
Code lines: 7
Contained by: file dark_matter.particle.warm_dark_matter.thermal.F90

function: wdmthermaldegreesoffreedomeffectivedecoupling
Description: Return the effective number of relativistic degrees of freedom at the time of decoupling of a thermal warm dark matter particle. The effective number of relativistic degrees of freedom at the time of decoupling is determined from the requirement that the warm dark matter particle have the correct relic density to provide the entire mass of dark matter [Hogan, 1999, eqn. 17].
Code lines: 10
Contained by: file dark_matter.particle.warm_dark_matter.thermal.F90

subroutine: wdmthermaldestructor
Description: Destructor for the cut off cooling rate class.
Code lines: 7

Contained by: file dark_matter.particle.warm_dark_matter.thermal.F90

function: wdmthermalmass
Description: Return the mass, in units of keV, of a thermal warm dark matter particle.
Code lines: 7
Contained by: file dark_matter.particle.warm_dark_matter.thermal.F90

file: dark_matter_halos.Correa2015.utilities.F90
Description: Contains a module which implements utility functions for the Correa et al. [2015] dark matter halo models.
Code lines: 53

module: dark_matter_halos_correa2015
Description: Implements utility functions for the Correa et al. [2015] dark matter halo models.
Code lines: 31
Contained by: file dark_matter_halos.Correa2015.utilities.F90
Used by: function correa2015time function correa2015concentration

subroutine: dark_matter_halo_correa2015_fit_parameters
Description: Computes fitting function parameters for the Correa et al. [2015] dark matter halo models.
Code lines: 21
Contained by: module dark_matter_halos_correa2015
Modules used: cosmological_density_field linear_growth numerical_constants_math

file: dark_matter_halos.formation_times.F90
Description: Contains a module which implements calculations of dark matter halo formation times.
Code lines: 64

module: dark_matter_halo_formation_times
Description: Implements calculations of dark matter halo formation times.
Code lines: 42
Contained by: file dark_matter_halos.formation_times.F90
Used by: function zhao2009concentration subroutine node_component_merging_statistics_standard_merger_tree_init

function: dark_matter_halo_formation_time
Description: Returns the time at which the main branch progenitor of node first had a mass equal to formationMassFraction of the current mass.
Code lines: 32
Contained by: module dark_matter_halo_formation_times
Modules used: dark_matter_halo_mass_accretion_histories galacticus_nodes

file: dark_matter_halos.mass_accretion_history.Correa2015.F90
Description: An implementation of dark matter halo mass accretion histories using the Correa et al. [2015] algorithm.
Code lines: 143
Modules used: cosmological_density_field cosmology_functions linear_growth

function: `correa2015constructorinternal`
Description: Generic constructor for the `correa2015` dark matter halo mass accretion history class.
Code lines: 10
Contained by: file `dark_matter_halos.mass_accretion_history.Correa2015.F90`

function: `correa2015constructorparameters`
Description: Constructor for the `correa2015` dark matter halo mass accretion history class which takes a parameter set as input.
Code lines: 20
Contained by: file `dark_matter_halos.mass_accretion_history.Correa2015.F90`
Modules used: `input_parameters`

subroutine: `correa2015destructor`
Description: Destructor for the `correa2015` dark matter halo mass accretion history class.
Code lines: 9
Contained by: file `dark_matter_halos.mass_accretion_history.Correa2015.F90`

function: `correa2015time`
Description: Compute the time corresponding to `mass` in the mass accretion history of `thisNode` using the algorithm of Correa et al. [2015].
Code lines: 50
Contained by: file `dark_matter_halos.mass_accretion_history.Correa2015.F90`
Modules used: `dark_matter_halos_correa2015` `galacticus_nodes` `root_finder`

function: `redshiftmasssolver`
Description: Root solver function used in finding the redshift corresponding to a given mass in the Correa et al. [2015] mass accretion history algorithm.
Code lines: 8
Contained by: function `correa2015time`

interface: `darkmatterhalomassaccretionhistorycorrea2015`
Description: Constructors for the `correa2015` dark matter halo mass accretion history class.
Code lines: 4
Contained by: file `dark_matter_halos.mass_accretion_history.Correa2015.F90`

file: `dark_matter_halos.mass_accretion_history.F90`
Description: Contains a module which provides a class for calculations of dark matter halo mass accretion histories.
Code lines: 41

module: `dark_matter_halo_mass_accretion_histories`
Description: Provides a class for calculations of dark matter halo mass accretion histories.
Code lines: 19
Contained by: file `dark_matter_halos.mass_accretion_history.F90`
Modules used: `galacticus_nodes`
Used by: function `dark_matter_halo_formation_time` file `dark_matter_profiles.structure.concentration.Zhao2009.F90`

subroutine galacticus_function_classes_destroy
subroutine galacticus_state_retrieve
subroutine galacticus_state_store
file merger_trees.construct.smooth_accretion.F90
subroutine node_component_mass_flow_statistics_standard_merger_tree_init
module node_component_merging_statistics_standard
program test_zhao2009_flat
program test_zhao2009_dark_energy
program test_zhao2009_open
program test_correa2015_mah

file: dark_matter_halos.mass_accretion_history.Wechsler2002.F90
Description: An implementation of dark matter halo mass accretion histories using the Wechsler et al. [2002] algorithm.
Code lines: 168
Modules used: cosmological_density_field cosmology_functions

interface: darkmatterhalomassaccretionhistorywechsler2002
Description: Constructors for the wechsler2002 dark matter halo mass accretion history class.
Code lines: 4
Contained by: file dark_matter_halos.mass_accretion_history.Wechsler2002.F90

function: wechsler2002constructorinternal
Description: Generic constructor for the wechsler2002 dark matter halo mass accretion history class.
Code lines: 14
Contained by: file dark_matter_halos.mass_accretion_history.Wechsler2002.F90
Modules used: galacticus_error

function: wechsler2002constructorparameters
Description: Default constructor for the wechsler2002 dark matter halo mass accretion history class.
Code lines: 42
Contained by: file dark_matter_halos.mass_accretion_history.Wechsler2002.F90
Modules used: input_parameters

subroutine: wechsler2002destructor
Description: Destructor for the wechsler2002 dark matter halo mass accretion history class.
Code lines: 9
Contained by: file dark_matter_halos.mass_accretion_history.Wechsler2002.F90

function: wechsler2002time
Description: Compute the time corresponding to mass in the mass accretion history of thisNode using the algorithm of Wechsler et al. [2002].
Code lines: 48
Contained by: file dark_matter_halos.mass_accretion_history.Wechsler2002.F90
Modules used: galacticus_nodes

function: expansionfactoratformation
Description: Computes the expansion factor at formation using the simple model of Bullock et al. [2001].
Code lines: 16
Contained by: function wechsler2002time

file: dark_matter_halos.mass_accretion_history.Zhao2009.F90
Description: An implementation of dark matter halo mass accretion histories using the Zhao et al. [2009] algorithm.
Code lines: 167
Modules used: cosmological_density_field

interface: darkmatterhalomassaccretionhistoryzhao2009
Description: Constructors for the zhao2009 dark matter halo mass accretion history class.
Code lines: 4
Contained by: file dark_matter_halos.mass_accretion_history.Zhao2009.F90

function: zhao2009constructorinternal
Description: Generic constructor for the zhao2009 dark matter halo mass accretion history class.
Code lines: 9
Contained by: file dark_matter_halos.mass_accretion_history.Zhao2009.F90

function: zhao2009constructorparameters
Description: Constructor for the zhao2009 dark matter halo mass accretion history class which takes a parameter set as input.
Code lines: 17
Contained by: file dark_matter_halos.mass_accretion_history.Zhao2009.F90
Modules used: input_parameters

subroutine: zhao2009destructor
Description: Destructor for the zhao2009 dark matter halo mass accretion history class.
Code lines: 8
Contained by: file dark_matter_halos.mass_accretion_history.Zhao2009.F90

function: zhao2009time
Description: Compute the time corresponding to mass in the mass accretion history of thisNode using the algorithm of Zhao et al. [2009].
Code lines: 82
Contained by: file dark_matter_halos.mass_accretion_history.Zhao2009.F90
Modules used: fgsl, galacticus_error, galacticus_nodes, ode_solver

function: growthrateodes
Description: System of differential equations to solve for the growth rate.
Code lines: 31
Contained by: function zhao2009time

file: dark_matter_halos.mass_loss_rates.F90
Description: Contains a module which implements calculations of mass loss rates from dark matter halos.
Code lines: 41

module: dark_matter_halos_mass_loss_rates
Description: Implements calculations of mass loss rates from dark matter halos.
Code lines: 19
Contained by: file dark_matter_halos.mass_loss_rates.F90

Modules used: galacticus_nodes
Used by: subroutine galacticus_function_classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; module node_component_satellite_standard

file: dark_matter_halos.mass_loss_rates.vanDenBosch.F90
Description: Implementation of mass loss rates from dark matter halos using the prescription of van den Bosch et al. [2005].
Code lines: 130
Modules used: cosmology_functions; virial_density_contrast

interface: darkmatterhalomasslossratevandenbosch
Description: Constructors for the vanDenBosch dark matter halo mass loss rate class.
Code lines: 4
Contained by: file dark_matter_halos.mass_loss_rates.vanDenBosch.F90

function: vandenboschconstructorinternal
Description: Internal constructor for the vanDenBosch dark matter halo mass loss rate class.
Code lines: 10
Contained by: file dark_matter_halos.mass_loss_rates.vanDenBosch.F90

function: vandenboschconstructorparameters
Description: Constructor for the vanDenBosch dark matter halo mass loss rate class which builds the object from a parameter set.
Code lines: 33
Contained by: file dark_matter_halos.mass_loss_rates.vanDenBosch.F90
Modules used: input_parameters

subroutine: vandenboschdestructor
Description: Destructor for the vanDenBosch dark matter halo mass loss rate class.
Code lines: 8
Contained by: file dark_matter_halos.mass_loss_rates.vanDenBosch.F90

function: vandenboschrate
Description: Returns the mass loss rate from the dark matter halo of the given node in units of $M_\odot$/Gyr.
Code lines: 23
Contained by: file dark_matter_halos.mass_loss_rates.vanDenBosch.F90
Modules used: galacticus_nodes

file: dark_matter_halos.mass_loss_rates.zero.F90
Description: Implementation of zero mass loss rate from dark matter halos.
Code lines: 62

interface: darkmatterhalomasslossratezero
Description: Constructors for the zero dark matter halo mass loss rate class.
Code lines: 3
Contained by: file dark_matter_halos.mass_loss_rates.zero.F90

function: zeroconstructorparameters

Description: Constructor for the zero dark matter halo mass loss rate class which builds the object from a parameter set.
Code lines: 10
Contained by: file dark_matter_halos.mass_loss_rates.zero.F90
Modules used: input_parameters

function: zerorate
Description: Returns the mass loss rate from the dark matter halo of the given node in units of $M_\odot$/Gyr.
Code lines: 9
Contained by: file dark_matter_halos.mass_loss_rates.zero.F90

file: dark_matter_halos.scales.F90
Description: Contains a module which provides a class implementing scales of dark matter halo scales.
Code lines: 89

module: dark_matter_halo_scales
Description: Provides a class implementing scales of dark matter halo scales.
Code lines: 67
Contained by: file dark_matter_halos.scales.F90
Modules used: galacticus_nodes
Used by:
file accretion.halo.simple.F90
file black_holes.binaries.initial_-separation.Volonteri_2003.F90
file black_holes.binaries.separation_-growth_rate.standard.F90
file cooling.cold_mode.infall_-rate.dynamical_time.F90
file cooling.cooling_radius.beta_-profile.F90
file cooling.cooling_-radius.isothermal_profile.F90
function isothermalradiusgrowthrate
file cooling.cooling_-rate.White-Frenk.F90
file cooling.cooling_rate.cut_off.F90
file cooling.time_-available.White-Frenk.F90
file dark_matter_-halos.spins.distributions.N-body_-errors.F90
file dark_matter_profiles.adiabatic_-Gnedin2004.F90
module dark_matter_profiles_generic
function bullock2001constructorinternal
function correa2015constructorinternal
function diemerkravtsov2014constructorinternal
function diemerkravtsov2014darkmatterprofiledefinition
subroutine duttonmaccio2014definitions
function gao2008constructorinternal
function klypin2015constructorinternal
function ludlow2016fitconstructorinternal
function munozcuartas2011constructorinternal
function nfw1996constructorinternal
function prada2011constructorinternal
function schneider2015darkmatterprofiledefinition
function zhao2009constructorinternal
file dark_matter_-profiles.structure.scale.concentration.F90
module dark_matter_profiles_dmo

function galactic_structure_radius_enclosing_density
function galactic_structure_potential

file galactic_structure.radius_solver.linear.F90
function hivshalomassrelationpadmanabhan2017constdistributionmaBett2007.F90
subroutine galacticus_state_retrieve
subroutine halo_model_projected_correlation
file hot_halo.mass_distribution.PatejLoeb2015.F90
file hot_halo.mass_distribution.cored.core_radius.growing.F90
file hot_halo.outflow_reincorporation.Henriques2013.F90

file hot_halo.ram_pressure_stripping.Font2008.F90

file hot_halo.ram_pressure_stripping.timescale.ram_pressure_acceleration.F90
file hot_halo.temperature_profile.virial.F90

file merger_trees.construct.read.F90

subroutine merger_tree_structure_output
file models.likelihoods.halo_mass_function.F90
file models.likelihoods.spin_distribution.F90
file nodes.property_extractor.ICM_Xray_luminosity.F90
file nodes.property_extractor.density_contrasts.F90
file nodes.property_extractor.rotation_curve.F90
file nodes.property_extractor.velocity_dispersion.F90
module node_component_black_hole_noncentral

function galactic_structure_radius_enclosing_mass
file galactic_structure.radius_solver.fixed.F90
subroutine galacticus_function_classes_destroy
file galacticus.output.analyses.spin_distribution.Bett2007.F90
subroutine galacticus_state_store
file hot_halo.cold_mode.density_profile.core_radius.virial_radius.F90
file hot_halo.mass_distribution.Ricotti2000.F90
file hot_halo.mass_distribution.cored.core_radius.virial_radius_fraction.F90
file hot_halo.outflow_reincorporation.halo_dynamical_time.F90
file hot_halo.ram_pressure_stripping.timescale.halo_dynamical_time.F90
file hot_halo.ram_pressure_stripping.virial_radius.F90

file merger_trees.construct.build.masses.fixed_mass.F90
file merger_trees.operators.particulate.F90
subroutine merger_trees_render_dump
file models.likelihoods.projected_correlation_function.F90
file nodes.property_extractor.ICM_SZ.F90
file nodes.property_extractor.density.F90
file nodes.property_extractor.projected_density.F90
file nodes.property_extractor.spin_Bullock.F90
file nodes.property_extractor.virial_properties.F90
module node_component_black_hole_simple

module node_component_black_hole_standard
module node_component_dark_matter_profile_scale
subroutine node_component_dark_matter_profile_scale_preset_rate_compute
module node_component_disk_standard
module node_component_disk_very_simple
subroutine node_component_disk_very_simple_rate_compute
module node_component_dynamics_statistics_bars
module node_component_hot_halo_cold_mode
subroutine node_component_hot_halo_cold_mode_formation
subroutine node_component_hot_halo_cold_mode_promote
module node_component_hot_halo_outflow_tracking
subroutine node_component_hot_halo_outflow_tracking_scale_set
module node_component_hot_halo_standard
module node_component_hot_halo_very_simple
module node_component_merging_statistics_recent
module node_component_position_trace_dark_matter
module node_component_satellite_orbiting
module node_component_spheroid_standard
subroutine node_component_spheroid_standard_radius_solver_plausibility
module node_component_spheroid_very_simple
module node_component_spin_vitvitska
file satellites.dynamical_friction.acceleration.Chandrasekhar1943.F90
file satellites.merging.remnant_sizes.Covington2008.F90
file satellites.merging.timescale.Boylan-Kolchin2008.F90
file satellites.merging.timescale.Jiang2008.F90
file satellites.merging.timescale.Lacey-Cole.F90
file satellites.merging.virial_orbits.Benson2005.F90
file satellites.merging.virial_orbits.Jiang2014.F90
file satellites.merging.virial_orbits.Wetzel2010.F90
file satellites.merging.virial_orbits.fixed.F90
function satellite_orbit_equivalent_circular_orbit_radius
file satellites.orphans.distributions.trace_dark_matter.F90
file satellites.tidal_heating.rate.Gnedin1999.F90
file satellites.tidal_stripping.rate.Zentner2005.F90
function zentner2005masslossrate
file satellites.tidal_stripping.rate.zero.F90
file star_formation.feedback.disks.halo_scaling.F90
file star_formation.timescales.disks.halo_scaling.F90
file statistics.Nbody.halos.mass_errors.SO_halo_finder.F90
file structure_formation.halo_mass_function.friends_of_friends_bias.F90
module virial_density_contrast_percolation_utilities
file tasks.halo_mass_function.F90

file tasks.halo_model.projected_correlation_function.F90
program test_dark_matter_halo_radius_enclosing_mass
program test_dark_matter_profiles
program test_dark_matter_profiles_generic
program test_dark_matter_profiles_heated

file: dark_matter_halos.scales.virial_density_contrast.F90
Description: An implementation of dark matter halo scales based on virial density contrast.
Code lines: 500
Modules used: cosmology_functions, cosmology_parameters, kind_numbers, tables, virial_density_contrast

interface: darkmatterhaloscalevirialdensitycontrastdefinition
Description: Constructors for the virialDensityContrastDefinition dark matter halo scales class.
Code lines: 4
Contained by: file dark_matter_halos.scales.virial_density_contrast.F90

subroutine: virialdensitycontrastdefinitionautohook
Description: Attach to the calculation reset event.
Code lines: 8
Contained by: file dark_matter_halos.scales.virial_density_contrast.F90
Modules used: events_hooks

subroutine: virialdensitycontrastdefinitioncalculationreset
Description: Reset the halo scales calculation.
Code lines: 12
Contained by: file dark_matter_halos.scales.virial_density_contrast.F90

subroutine: virialdensitycontrastdefinitiondeepcopy
Description: Perform a deep copy of the object.
Code lines: 56
Contained by: file dark_matter_halos.scales.virial_density_contrast.F90
Modules used: galacticus_error

subroutine: virialdensitycontrastdefinitiondeepcopyassign
Description: Perform pointer assignment during a deep copy of the object.
Code lines: 11
Contained by: file dark_matter_halos.scales.virial_density_contrast.F90

subroutine: virialdensitycontrastdefinitiondestructor
Description: Destructir for the virialDensityContrastDefinition dark matter halo scales class.
Code lines: 12
Contained by: file dark_matter_halos.scales.virial_density_contrast.F90
Modules used: events_hooks

function: virialdensitycontrastdefinitiondynamicaltimescale
Description: Returns the dynamical timescale for node.

Code lines: 22
Contained by: file dark_matter_halos.scales.virial_density_contrast.F90
Modules used: numerical_constants_astronomical

function: virialdensitycontrastdefinitioninternal
Description: Default constructor for the virialDensityContrastDefinition dark matter halo scales class.
Code lines: 33
Contained by: file dark_matter_halos.scales.virial_density_contrast.F90
Modules used: galacticus_error

function: virialdensitycontrastdefinitionmeandensity
Description: Returns the mean density for node.
Code lines: 46
Contained by: file dark_matter_halos.scales.virial_density_contrast.F90
Modules used: galacticus_nodes

function: virialdensitycontrastdefinitionmeandensitygrowthrate
Description: Returns the growth rate of the mean density for node.
Code lines: 36
Contained by: file dark_matter_halos.scales.virial_density_contrast.F90
Modules used: galacticus_nodes

function: virialdensitycontrastdefinitionparameters
Description: Constructor for the virialDensityContrastDefinition dark matter halo scales class which takes a parameter set as input.
Code lines: 21
Contained by: file dark_matter_halos.scales.virial_density_contrast.F90
Modules used: input_parameters

function: virialdensitycontrastdefinitionvirialradius
Description: Returns the virial radius scale for node.
Code lines: 29
Contained by: file dark_matter_halos.scales.virial_density_contrast.F90
Modules used: galacticus_nodes math_exponentiation numerical_constants_math

function: virialdensitycontrastdefinitionvirialradiusgradientlogmass
Description: Returns the logarithmic gradient of virial radius with halo mass at fixed epoch for node.
Code lines: 16
Contained by: file dark_matter_halos.scales.virial_density_contrast.F90
Modules used: numerical_constants_math

function: virialdensitycontrastdefinitionvirialradiusgrowthrate
Description: Returns the growth rate of the virial radius scale for node.
Code lines: 17
Contained by: file dark_matter_halos.scales.virial_density_contrast.F90
Modules used: galacticus_nodes

function: `virialdensitycontrastdefinitionvirialtemperature`
Description: Returns the virial temperature (in Kelvin) for `node`.
Code lines: 23
Contained by: file dark_matter_halos.scales.virial_density_contrast.F90
Modules used: numerical_constants_astronomical numerical_constants_physical

function: `virialdensitycontrastdefinitionvirialvelocity`
Description: Returns the virial velocity scale for `node`.
Code lines: 28
Contained by: file dark_matter_halos.scales.virial_density_contrast.F90
Modules used: galacticus_nodes numerical_constants_physical

function: `virialdensitycontrastdefinitionvirialvelocitygrowthrate`
Description: Returns the growth rate of the virial velocity scale for `node`.
Code lines: 17
Contained by: file dark_matter_halos.scales.virial_density_contrast.F90
Modules used: galacticus_nodes

file: `dark_matter_halos.spins.F90`
Description: Contains a module which implements calculations of dark matter halo angular momentum.
Code lines: 106

module: `dark_matter_halo_spins`
Description: Implements calculations of dark matter halo angular momentum.
Code lines: 84
Contained by: file dark_matter_halos.spins.F90
Used by: function spinbullockextract
subroutine node_component_disk_very_simple_size_radius_solver
subroutine node_component_hot_halo_cold_mode_node_merger
subroutine node_component_hot_halo_cold_mode_rate_compute
subroutine node_component_hot_halo_cold_mode_tree_initialize
subroutine node_component_hot_halo_standard_node_merger
subroutine node_component_hot_halo_standard_rate_compute
subroutine node_component_hot_halo_standard_tree_initialize
subroutine node_component_spheroid_very_simple_radius_solver
module node_component_spin_vitvitska

subroutine: `assertpropertiesgettable`
Description: Assert that properties required for spin calculations are gettable.
Code lines: 19
Contained by: module dark_matter_halo_spins
Modules used: galacticus_error galacticus_nodes
iso_varying_string

function: `dark_matter_halo_angular_momentum`
Description: Returns the total anuglar momentum of `node` based on its mass, energy and spin parameter.
Code lines: 16

Contained by: module dark_matter_halo_spins
Modules used: dark_matter_profiles_dmo galacticus_nodes numerical_constants_physical

function: dark_matter_halo_angular_momentum_growth_rate
Description: Returns the rate of change of the total anuglar momentum of node based on its mass, energy and spin parameter.
Code lines: 15
Contained by: module dark_matter_halo_spins
Modules used: dark_matter_profiles_dmo galacticus_nodes

function: dark_matter_halo_spin
Description: Returns the spin of node given its angular momentum.
Code lines: 15
Contained by: module dark_matter_halo_spins
Modules used: dark_matter_profiles_dmo galacticus_nodes numerical_constants_physical

file: dark_matter_halos.spins.distributions.Bett2007.F90
Description: An implementation of the dark matter halo spin distribution which uses the fitting function proposed by Bett et al. [2007].
Code lines: 163
Modules used: table_labels tables

function: bett2007constructorinternal
Description: Internal constructor for the bett2007 dark matter halo spin distribution class.
Code lines: 31
Contained by: file dark_matter_halos.spins.distributions.Bett2007.F90
Modules used: array_utilities gamma_functions iso_c_binding

function: bett2007constructorparameters
Description: Constructor for the bett2007 dark matter halo spin distribution class which takes a parameter list as input.
Code lines: 31
Contained by: file dark_matter_halos.spins.distributions.Bett2007.F90
Modules used: input_parameters

subroutine: bett2007destructor
Description: Destructor for the bett2007 dark matter halo spin distribution class.
Code lines: 9
Contained by: file dark_matter_halos.spins.distributions.Bett2007.F90

function: bett2007distribution
Description: Compute the spin parameter distribution for the given node assuming the fitting function of Bett et al. [2007].
Code lines: 12
Contained by: file dark_matter_halos.spins.distributions.Bett2007.F90
Modules used: galacticus_nodes

function: `bett2007sample`
Description: Sample from a Bett et al. [2007] spin parameter distribution for the given `node`.
Code lines: 16
Contained by: file `dark_matter_halos.spins.distributions.Bett2007.F90`
Modules used: `galacticus_error` `pseudo_random`

interface: `halospindistributionbett2007`
Description: Constructors for the `bett2007` dark matter halo spin distribution class.
Code lines: 5
Contained by: file `dark_matter_halos.spins.distributions.Bett2007.F90`

file: `dark_matter_halos.spins.distributions.F90`
Description: Contains a module that provides a class for dark matter halo spin distributions.
Code lines: 46

module: `halo_spin_distributions`
Description: Provides a class for dark matter halo spin distributions.
Code lines: 24
Contained by: file `dark_matter_halos.spins.distributions.F90`
Modules used: `galacticus_nodes`
Used by:
- subroutine `galacticus_function_classes_destroy`
- file `galacticus.output.analyses.distribution_operator.spin_N-body_errors.F90`
- function `spindistributionbett2007constructorinternal`
- subroutine `galacticus_state_retrieve`
- subroutine `galacticus_state_store`
- file `merger_trees.construct.read.F90`
- function `spindistributionevaluate`
- module `node_component_spin_vitvitska`
- module `node_component_spin_random`
- file `tasks.halo_spin_distribution.F90`
- subroutine `halospindistributionperform`

file: `dark_matter_halos.spins.distributions.N-body_errors.F90`
Description: An implementation of the dark matter halo spin distribution which modifies another spin distribution to account for the effects of particle noise errors in spins measured in N-body simulations.
Code lines: 684
Modules used: `cosmology_functions` `dark_matter_halo_scales` `dark_matter_profile_scales` `dark_matter_profiles_dmo` `halo_mass_functions` `statistics_nbody_halo_mass_errors`

interface: `halospindistributionnbodyerrors`
Description: Constructors for the `nbodyErrors` dark matter halo spin distribution class.
Code lines: 5
Contained by: file `dark_matter_halos.spins.distributions.N-body_errors.F90`

function: `nbodyerrorsconstructorinternal`
Description: Internal constructor for the `nbodyErrors` dark matter halo spin distribution class.
Code lines: 22
Contained by: file `dark_matter_halos.spins.distributions.N-body_errors.F90`

function: `nbodyerrorsconstructorparameters`
Description: Constructor for the `nbodyErrors` dark matter halo spin distribution class which takes a parameter list as input.
Code lines: 70
Contained by: file `dark_matter_halos.spins.distributions.N-body_errors.F90`
Modules used: `input_parameters`

subroutine: `nbodyerrorsdestructor`
Description: Destructor for the `nbodyErrors` dark matter halo spin distribution class.
Code lines: 12
Contained by: file `dark_matter_halos.spins.distributions.N-body_errors.F90`

function: `nbodyerrorsdistribution`
Description: Compute the spin distribution.
Code lines: 30
Contained by: file `dark_matter_halos.spins.distributions.N-body_errors.F90`
Modules used: `galacticus_nodes`

function: `nbodyerrorsdistributionaveraged`
Description: Compute the spin distribution averaged over all halos more massive than the given `massLimit`.
Code lines: 34
Contained by: file `dark_matter_halos.spins.distributions.N-body_errors.F90`
Modules used: `galacticus_calculations_resets` `galacticus_nodes`

function: `nbodyerrorsdistributionfixedpoint`
Description: Compute the spin distribution for a fixed point in intrinsic mass and spin.
Code lines: 27
Contained by: file `dark_matter_halos.spins.distributions.N-body_errors.F90`
Modules used: `galacticus_nodes`

function: `nbodyerrorsnoncentralchisquaremode`
Description: Computes the mode of the degree-3 non-central chi-squared distribution function for given χ.
Code lines: 13
Contained by: file `dark_matter_halos.spins.distributions.N-body_errors.F90`
Modules used: `root_finder`

function: `nbodyerrorsnoncentralchisquaremoderoot`
Description: Root function used in finding the mode of the degree-3 non-central chi-squared distribution function.
Code lines: 7
Contained by: file `dark_matter_halos.spins.distributions.N-body_errors.F90`

function: `nbodyerrorssample`
Description: Sample from the halo spin distribution.
Code lines: 11
Contained by: file `dark_matter_halos.spins.distributions.N-body_errors.F90`

Modules used: galacticus_error

subroutine: nbodyerrorstabulate
Description: Tabulate the halo spin distribution.
Code lines: 339
Contained by: file dark_matter_halos.spins.distributions.N-body_errors.F90
Modules used: fgsl, galacticus_calculations_resets, galacticus_error, galacticus_nodes, iso_c_binding, memory_management, numerical_constants_math, numerical_integration

function: errorsspindependent
Description: Compute the spin-dependent error term.
Code lines: 8
Contained by: subroutine nbodyerrorstabulate

function: massintegral
Description: Integral over the halo mass function and halo mass error distribution.
Code lines: 14
Contained by: subroutine nbodyerrorstabulate

function: massspinintegral
Description: Integral over the halo mass function, spin distribution, halo mass error distribution, and spin error distribution.
Code lines: 66
Contained by: subroutine nbodyerrorstabulate
Modules used: galacticus_error, input_parameters

function: productdistributionintegral
Description: Product distribution integrand.
Code lines: 39
Contained by: subroutine nbodyerrorstabulate

function: spinerrorintegral
Description: Integral over the spin error distribution.
Code lines: 35
Contained by: subroutine nbodyerrorstabulate

function: spinintegral
Description: Integral over the intrinsic spin distribution, and spin error distribution.
Code lines: 14
Contained by: subroutine nbodyerrorstabulate

file: dark_matter_halos.spins.distributions.delta_function.F90
Description: An implementation of the dark matter halo spin distribution which assumes a δ-function distribution.
Code lines: 114

function: deltafunctionconstructorinternal

Description: Internal constructor for the `deltaFunction` dark matter halo spin distribution class.
Code lines: 9
Contained by: file dark_matter_halos.spins.distributions.delta_function.F90

function: `deltafunctionconstructorparameters`
Description: Constructor for the `deltaFunction` dark matter halo spin distribution class which takes a parameter list as input.
Code lines: 21
Contained by: file dark_matter_halos.spins.distributions.delta_function.F90
Modules used: input_parameters

subroutine: `deltafunctiondestructor`
Description: Destructor for the `deltaFunction` dark matter halo spin distribution class.
Code lines: 9
Contained by: file dark_matter_halos.spins.distributions.delta_function.F90

function: `deltafunctiondistribution`
Description: Return the spin parameter distribution for the given `node`.
Code lines: 11
Contained by: file dark_matter_halos.spins.distributions.delta_function.F90
Modules used: galacticus_error

function: `deltafunctionsample`
Description: Sample from a δ-function spin parameter distribution for the given `node`.
Code lines: 10
Contained by: file dark_matter_halos.spins.distributions.delta_function.F90

interface: `halospindistributiondeltafunction`
Description: Constructors for the `deltaFunction` dark matter halo spin distribution class.
Code lines: 5
Contained by: file dark_matter_halos.spins.distributions.delta_function.F90

file: `dark_matter_halos.spins.distributions.lognormal.F90`
Description: An implementation of the dark matter halo spin distribution which assumes a log-normal distribution.
Code lines: 117

interface: `halospindistributionlognormal`
Description: Constructors for the `logNormal` dark matter halo spin distribution class.
Code lines: 5
Contained by: file dark_matter_halos.spins.distributions.lognormal.F90

function: `lognormalconstructorinternal`
Description: Internal constructor for the `logNormal` dark matter halo spin distribution class.
Code lines: 10
Contained by: file dark_matter_halos.spins.distributions.lognormal.F90

function: `lognormalconstructorparameters`
Description: Constructor for the `logNormal` dark matter halo spin distribution class which takes a parameter list as input.
Code lines: 32

Contained by: file dark_matter_halos.spins.distributions.lognormal.F90
Modules used: input_parameters

function: lognormaldistribution
Description: Return the spin parameter distribution for the given node assuming a log-normal distribution.
Code lines: 14
Contained by: file dark_matter_halos.spins.distributions.lognormal.F90
Modules used: galacticus_nodes numerical_constants_math

function: lognormalsample
Description: Sample from a log-normal spin parameter distribution for the given node.
Code lines: 10
Contained by: file dark_matter_halos.spins.distributions.lognormal.F90

file: dark_matter_profiles.F90
Description: Contains a module which provides an object that implements non-dark-matter-only dark matter halo profiles.
Code lines: 200

module: dark_matter_profiles
Description: Provides an object that implements non-dark-matter-only dark matter halo profiles.
Code lines: 178
Contained by: file dark_matter_profiles.F90
Modules used: dark_matter_profiles_generic galacticus_nodes kind_numbers
Used by: module dark_matter_profile_structure_tasks; function galactic_structure_radius_enclosing_mass; file galactic_structure.radius_solver.equilibrium.F90; subroutine galacticus_function_classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; program test_dark_matter_profiles_generic

file: dark_matter_profiles.adiabatic_Gnedin2004.F90
Description: An implementation of adiabaticGnedin2004 dark matter halo profiles.
Code lines: 924
Modules used: cosmology_parameters dark_matter_halo_scales dark_matter_profiles_dmo dark_matter_profiles_generic galactic_structure_options math_exponentiation

subroutine: adiabaticgnedin2004autohook
Description: Attach to the calculation reset event.
Code lines: 8
Contained by: file dark_matter_profiles.adiabatic_Gnedin2004.F90
Modules used: events_hooks

subroutine: adiabaticgnedin2004calculationreset
Description: Reset the dark matter profile calculation.

Code lines: 13
Contained by: file dark_matter_profiles.adiabatic_Gnedin2004.F90

function: adiabaticgnedin2004circularvelocity
Description: Returns the circular velocity (in km/s) in the dark matter profile of node at the given radius (given in units of Mpc).
Code lines: 14
Contained by: file dark_matter_profiles.adiabatic_Gnedin2004.F90

function: adiabaticgnedin2004circularvelocitymaximum
Description: Returns the maximum circular velocity (in km/s) in the dark matter profile of node.
Code lines: 12
Contained by: file dark_matter_profiles.adiabatic_Gnedin2004.F90

subroutine: adiabaticgnedin2004computefactors
Description: Compute various factors needed when solving for the initial radius in the dark matter halo using the adiabatic contraction algorithm of Gnedin et al. [2004].
Code lines: 183
Contained by: file dark_matter_profiles.adiabatic_Gnedin2004.F90
Modules used: galacticus_nodes, hot_halo_mass_distributions, node_component_black_hole_simple_structure, node_component_black_hole_standard_structure_tasks, node_component_hot_halo_cold_mode_structure_tasks, numerical_constants_physical

function: adiabaticgnedin2004constructorinternal
Description: Generic constructor for the adiabaticGnedin2004 dark matter profile class.
Code lines: 21
Contained by: file dark_matter_profiles.adiabatic_Gnedin2004.F90
Modules used: galacticus_error

function: adiabaticgnedin2004constructorparameters
Description: Default constructor for the adiabaticGnedin2004 dark matter halo profile class.
Code lines: 47
Contained by: file dark_matter_profiles.adiabatic_Gnedin2004.F90
Modules used: input_parameters

function: adiabaticgnedin2004density
Description: Returns the density (in $M_\odot$ Mpc^{-3}) in the dark matter profile of node at the given radius (given in units of Mpc).
Code lines: 18
Contained by: file dark_matter_profiles.adiabatic_Gnedin2004.F90

function: adiabaticgnedin2004densitylogslope
Description: Returns the logarithmic slope of the density in the dark matter profile of node at the given radius (given in units of Mpc).
Code lines: 16
Contained by: file dark_matter_profiles.adiabatic_Gnedin2004.F90

function: adiabaticgnedin2004derivativesolver

Description: Root function used in finding the derivative of the initial radius in the dark matter halo when solving for adiabatic contraction.
Code lines: 17
Contained by: file dark_matter_profiles.adiabatic_Gnedin2004.F90
Modules used: numerical_constants_math

subroutine: adiabaticgnedin2004destructor
Description: Destructor for the adiabaticGnedin2004 dark matter halo profile class.
Code lines: 11
Contained by: file dark_matter_profiles.adiabatic_Gnedin2004.F90
Modules used: events_hooks

function: adiabaticgnedin2004enclosedmass
Description: Returns the enclosed mass (in $M_\odot$) in the dark matter profile of node at the given radius (given in units of Mpc).
Code lines: 10
Contained by: file dark_matter_profiles.adiabatic_Gnedin2004.F90

function: adiabaticgnedin2004energy
Description: Return the energy of a adiabaticGnedin2004 halo density profile.
Code lines: 12
Contained by: file dark_matter_profiles.adiabatic_Gnedin2004.F90

function: adiabaticgnedin2004energygrowthrate
Description: Return the rate of change of the energy of a adiabaticGnedin2004 halo density profile.
Code lines: 12
Contained by: file dark_matter_profiles.adiabatic_Gnedin2004.F90

function: adiabaticgnedin2004freefallradius
Description: Returns the freefall radius in the adiabaticGnedin2004 density profile at the specified time (given in Gyr).
Code lines: 14
Contained by: file dark_matter_profiles.adiabatic_Gnedin2004.F90

function: adiabaticgnedin2004freefallradiusincreaserate
Description: Returns the rate of increase of the freefall radius in the adiabaticGnedin2004 density profile at the specified time (given in Gyr).
Code lines: 14
Contained by: file dark_matter_profiles.adiabatic_Gnedin2004.F90

function: adiabaticgnedin2004kspace
Description: Returns the Fourier transform of the adiabaticGnedin2004 density profile at the specified waveNumber (given in Mpc^{-1}), using the expression given in Cooray and Sheth (2002; eqn. 81).
Code lines: 14
Contained by: file dark_matter_profiles.adiabatic_Gnedin2004.F90

function: adiabaticgnedin2004massenclosed
Description: Unary function returning the enclosed mass in a component. Suitable for mapping over components. Ignores the dark matter profile.

Code lines: 14
Contained by: file dark_matter_profiles.adiabatic_Gnedin2004.F90
Modules used: galacticus_nodes

function: adiabaticgnedin2004potential
Description: Returns the potential (in $(\mathrm{km/s})^2$) in the dark matter profile of node at the given radius (given in units of Mpc).
Code lines: 19
Contained by: file dark_matter_profiles.adiabatic_Gnedin2004.F90

function: adiabaticgnedin2004radialmoment
Description: Returns the radial moment of the density in the dark matter profile of node between the given radiusMinimum and radiusMaximum (given in units of Mpc).
Code lines: 15
Contained by: file dark_matter_profiles.adiabatic_Gnedin2004.F90

function: adiabaticgnedin2004radialvelocitydispersion
Description: Returns the radial velocity dispersion (in km/s) in the dark matter profile of node at the given radius (given in units of Mpc).
Code lines: 14
Contained by: file dark_matter_profiles.adiabatic_Gnedin2004.F90

function: adiabaticgnedin2004radiusenclosingdensity
Description: Returns the radius (in Mpc) in the dark matter profile of node which encloses the given density (given in units of $M_\odot/\mathrm{Mpc}^{-3}$).
Code lines: 14
Contained by: file dark_matter_profiles.adiabatic_Gnedin2004.F90

function: adiabaticgnedin2004radiusenclosingmass
Description: Returns the radius (in Mpc) in the dark matter profile of node which encloses the given mass (given in units of $M_\odot$).
Code lines: 14
Contained by: file dark_matter_profiles.adiabatic_Gnedin2004.F90

function: adiabaticgnedin2004radiusfromspecificangularmomentum
Description: Returns the radius (in Mpc) in node at which a circular orbit has the given specificAngularMomentum (given in units of km s^{-1} Mpc).
Code lines: 14
Contained by: file dark_matter_profiles.adiabatic_Gnedin2004.F90

function: adiabaticgnedin2004radiusinitial
Description: Compute the initial radius in the dark matter halo using the adiabatic contraction algorithm of Gnedin et al. [2004].
Code lines: 86
Contained by: file dark_matter_profiles.adiabatic_Gnedin2004.F90
Modules used: root_finder

function: adiabaticgnedin2004radiusinitialderivative
Description: Compute the derivative of the initial radius in the dark matter halo using the adiabatic contraction algorithm of Gnedin et al. [2004].

Code lines: 35
Contained by: file dark_matter_profiles.adiabatic_Gnedin2004.F90
Modules used: root_finder

function: adiabaticgnedin2004radiusorbitalmean
Description: Returns the orbit averaged radius for dark matter corresponding the given radius using the model of Gnedin et al. [2004].
Code lines: 9
Contained by: file dark_matter_profiles.adiabatic_Gnedin2004.F90

function: adiabaticgnedin2004radiusorbitalmeanderivative
Description: Returns the derivative of the orbit averaged radius for dark matter corresponding the given radius using the model of Gnedin et al. [2004].
Code lines: 9
Contained by: file dark_matter_profiles.adiabatic_Gnedin2004.F90

function: adiabaticgnedin2004rotationnormalization
Description: Return the normalization of the rotation velocity vs. specific angular momentum relation.
Code lines: 12
Contained by: file dark_matter_profiles.adiabatic_Gnedin2004.F90

function: adiabaticgnedin2004solver
Description: Root function used in finding the initial radius in the dark matter halo when solving for adiabatic contraction.
Code lines: 13
Contained by: file dark_matter_profiles.adiabatic_Gnedin2004.F90

function: adiabaticgnedin2004velocitycircularsquared
Description: Unary function returning the squared rotation curve in a component. Suitable for mapping over components.
Code lines: 13
Contained by: file dark_matter_profiles.adiabatic_Gnedin2004.F90
Modules used: galacticus_nodes

function: adiabaticgnedin2004velocitycircularsquaredgradient
Description: Unary function returning the squared rotation curve gradient in a component. Suitable for mapping over components.
Code lines: 13
Contained by: file dark_matter_profiles.adiabatic_Gnedin2004.F90
Modules used: galacticus_nodes

interface: darkmatterprofileadiabaticgnedin2004
Description: Constructors for the adiabaticGnedin2004 dark matter halo profile class.
Code lines: 4
Contained by: file dark_matter_profiles.adiabatic_Gnedin2004.F90

file: dark_matter_profiles.dark_matter_only.F90
Description: An implementation of non-dark-matter-only dark matter halo profiles which are unchanged from their dark-matter-only counterpart.
Code lines: 308

Modules used: cosmology_parameters dark_matter_profiles_dmo

function: darkmatteronlycircularvelocity
Description: Returns the circular velocity (in km/s) in the dark matter profile of node at the given radius (given in units of Mpc).
Code lines: 10
Contained by: file dark_matter_profiles.dark_matter_only.F90

function: darkmatteronlycircularvelocitymaximum
Description: Returns the maximum circular velocity (in km/s) in the dark matter profile of node.
Code lines: 8
Contained by: file dark_matter_profiles.dark_matter_only.F90

function: darkmatteronlyconstructorinternal
Description: Generic constructor for the darkMatterOnly dark matter profile class.
Code lines: 12
Contained by: file dark_matter_profiles.dark_matter_only.F90

function: darkmatteronlyconstructorparameters
Description: Constructor for the darkMatterOnly non-dark-matter-only dark matter halo profile class which takes a parameter set as input.
Code lines: 20
Contained by: file dark_matter_profiles.dark_matter_only.F90
Modules used: input_parameters

function: darkmatteronlydensity
Description: Returns the density (in $M_\odot$ Mpc^{-3}) in the dark matter profile of node at the given radius (given in units of Mpc).
Code lines: 10
Contained by: file dark_matter_profiles.dark_matter_only.F90

function: darkmatteronlydensitylogslope
Description: Returns the logarithmic slope of the density in the dark matter profile of node at the given radius (given in units of Mpc).
Code lines: 10
Contained by: file dark_matter_profiles.dark_matter_only.F90

subroutine: darkmatteronlydestructor
Description: Destructor for the darkMatterOnly dark matter halo profile class.
Code lines: 9
Contained by: file dark_matter_profiles.dark_matter_only.F90

function: darkmatteronlyenclosedmass
Description: Returns the enclosed mass (in $M_\odot$) in the dark matter profile of node at the given radius (given in units of Mpc).
Code lines: 10
Contained by: file dark_matter_profiles.dark_matter_only.F90

function: darkmatteronlyenergy

Description: Return the energy of the dark matter halo density profile.
Code lines: 8
Contained by: file dark_matter_profiles.dark_matter_only.F90

function: darkmatteronlyenergygrowthrate
Description: Return the rate of change of the energy of the dark matter halo density profile.
Code lines: 8
Contained by: file dark_matter_profiles.dark_matter_only.F90

function: darkmatteronlyfreefallradius
Description: Returns the freefall radius in the dark matter halo density profile at the specified `time` (given in Gyr).
Code lines: 10
Contained by: file dark_matter_profiles.dark_matter_only.F90

function: darkmatteronlyfreefallradiusincreaserate
Description: Returns the freefall radius in the dark matter halo density profile at the specified `time` (given in Gyr).
Code lines: 10
Contained by: file dark_matter_profiles.dark_matter_only.F90

function: darkmatteronlykspace
Description: Returns the Fourier transform of the dark matter halo density profile at the specified `waveNumber` (given in Mpc^{-1}).
Code lines: 12
Contained by: file dark_matter_profiles.dark_matter_only.F90
Modules used: galacticus_error

function: darkmatteronlypotential
Description: Returns the potential (in $(\mathrm{km/s})^2$) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).
Code lines: 11
Contained by: file dark_matter_profiles.dark_matter_only.F90

function: darkmatteronlyradialmoment
Description: Returns the density (in $M_\odot$ Mpc^{-3}) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).
Code lines: 11
Contained by: file dark_matter_profiles.dark_matter_only.F90

function: darkmatteronlyradialvelocitydispersion
Description: Returns the radial velocity dispersion (in km/s) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).
Code lines: 10
Contained by: file dark_matter_profiles.dark_matter_only.F90

function: darkmatteronlyradiusenclosingdensity
Description: Returns the radius (in Mpc) in the dark matter profile of `node` which encloses the given `density` (given in units of $M_\odot/\mathrm{Mpc}^{-3}$).
Code lines: 10

Contained by: file dark_matter_profiles.dark_matter_only.F90

function: darkmatteronlyradiusenclosingmass

Description: Returns the radius (in Mpc) in the dark matter profile of `node` which encloses the given `mass` (given in units of $M_\odot$).
Code lines: 10
Contained by: file dark_matter_profiles.dark_matter_only.F90

function: darkmatteronlyradiusfromspecificangularmomentum

Description: Returns the radius (in Mpc) in `node` at which a circular orbit has the given `specificAngularMomentum` (given in units of km s^{-1} Mpc).
Code lines: 10
Contained by: file dark_matter_profiles.dark_matter_only.F90

function: darkmatteronlyrotationnormalization

Description: Return the normalization of the rotation velocity vs. specific angular momentum relation.
Code lines: 8
Contained by: file dark_matter_profiles.dark_matter_only.F90

interface: darkmatterprofiledarkmatteronly

Description: Constructors for the `darkMatterOnly` non-dark-matter-only dark matter halo profile class.
Code lines: 4
Contained by: file dark_matter_profiles.dark_matter_only.F90

file: dark_matter_profiles.generic.F90

Description: Contains a module which implements a base class for dark matter profiles from which both dark-matter-only and non-dark-matter-only profiles inherit.
Code lines: 819

module: dark_matter_profiles_generic

Description: A base class for dark matter profiles from which both dark-matter-only and non-dark-matter-only profiles inherit. Implements numerical calculations of certain halo properties which are to be used as a fall-back option when no analytical solution exists.
Code lines: 795
Contained by: file dark_matter_profiles.generic.F90
Modules used: dark_matter_halo_scales, function_classes, galacticus_nodes
Used by: module dark_matter_profiles; file dark_matter_profiles.adiabatic_Gnedin2004.F90; module dark_matter_profiles_dmo; file dark_matter_profiles_DMO.heated.F90; file dark_matter_profiles_DMO.truncated.F90; file dark_matter_profiles_DMO.truncated.exponential.F90; program test_dark_matter_halo_radius_enclosing_mass; program test_dark_matter_profiles_generic; program test_dark_matter_profiles_heated

type: darkmatterprofilegeneric

Description: A dark matter halo profile class implementing numerical calculations for generic dark matter halos.
Code lines: 150
Contained by: module dark_matter_profiles_generic

function: genericcircularvelocitymaximumnumerical
Description: Returns the maximum circular velocity (in km/s) in the dark matter profile of node.
Code lines: 16
Contained by: module dark_matter_profiles_generic
Modules used: root_finder

function: genericcircularvelocitynumerical
Description: Returns the circular velocity (in km/s) in the dark matter profile of node at the given radius (given in units of Mpc).
Code lines: 15
Contained by: module dark_matter_profiles_generic
Modules used: numerical_constants_physical

function: genericdensityevaluate
Description: GSL-callable function to evaluate the density of the dark matter profile.
Code lines: 7
Contained by: module dark_matter_profiles_generic

function: genericdensitylogslopenumerical
Description: Returns the logarithmic slope of the density in the dark matter profile of node at the given radius (given in units of Mpc).
Code lines: 16
Contained by: module dark_matter_profiles_generic
Modules used: numerical_differentiation

function: genericenclosedmassdifferencenumerical
Description: Returns the enclosed mass difference (in $M_\odot$) in the dark matter profile of node between the given radiusLower and radiusUpper (given in units of Mpc) using a numerical calculation.
Code lines: 33
Contained by: module dark_matter_profiles_generic
Modules used: fgsl numerical_integration

function: genericmassintegrand
Description: Integrand for mass in generic dark matter profiles.
Code lines: 12
Contained by: function genericenclosedmassdifferencenumerical
Modules used: numerical_constants_math

function: genericenclosedmassnumerical
Description: Returns the enclosed mass (in $M_\odot$) in the dark matter profile of node at the given radius (given in units of Mpc).
Code lines: 10
Contained by: module dark_matter_profiles_generic

function: genericenergyevaluate

Description: GSL-callable function to evaluate the energy of the dark matter profile.
Code lines: 16
Contained by: module dark_matter_profiles_generic
Modules used: functions_global

function: genericenergygrowthratenumerical
Description: Returns the rate of growth of the energy if the dark matter density profile.
Code lines: 32
Contained by: module dark_matter_profiles_generic
Modules used: numerical_differentiation

function: genericenergynumerical
Description: Return the energy of a generic dark matter density profile.
Code lines: 66
Contained by: module dark_matter_profiles_generic
Modules used: fgsl numerical_constants_math numerical_constants_physical numerical_integration

function: integrandenergykinetic
Description: Integrand for kinetic energy of the halo.
Code lines: 11
Contained by: function genericenergynumerical

function: integrandenergypotential
Description: Integrand for potential energy of the halo.
Code lines: 11
Contained by: function genericenergynumerical

function: integrandpseudopressure
Description: Integrand for pseudo-pressure ($\rho(r)\sigma^2(r)$) of the halo.
Code lines: 11
Contained by: function genericenergynumerical

function: genericfreefallradiusevaluate
Description: GSL-callable function to evaluate the freefall radius of the dark matter profile.
Code lines: 7
Contained by: module dark_matter_profiles_generic

function: genericfreefallradiusincreaseratenumerical
Description: Returns the rate of increase of the freefall radius in the dark matter density profile at the specified time (given in Gyr).
Code lines: 16
Contained by: module dark_matter_profiles_generic
Modules used: numerical_differentiation

function: genericfreefallradiusnumerical
Description: Returns the freefall radius in the adiabaticGnedin2004 density profile at the specified time (given in Gyr).
Code lines: 19
Contained by: module dark_matter_profiles_generic

Modules used: root_finder

function: genericspacenumerical
Description: Returns the Fourier transform of the dark matter density profile at the specified waveNumber (given in Mpc^{-1}).
Code lines: 34
Contained by: module dark_matter_profiles_generic
Modules used: fgsl numerical_integration

function: integrandfouriertransform
Description: Integrand for Fourier transform of the generic dark matter profile.
Code lines: 12
Contained by: function genericspacenumerical
Modules used: numerical_constants_math

function: genericpotentialdifferencenumerical
Description: Returns the potential difference (in $(\mathrm{km/s})^2$) in the dark matter profile of node between the given radiusLower and radiusUpper (given in units of Mpc) using a numerical calculation.
Code lines: 19
Contained by: module dark_matter_profiles_generic
Modules used: fgsl galactic_structure_options numerical_integration

function: genericpotentialnumerical
Description: Returns the potential (in $(\mathrm{km/s})^2$) in the dark matter profile of node at the given radius (given in units of Mpc) using a numerical calculation.
Code lines: 23
Contained by: module dark_matter_profiles_generic
Modules used: fgsl galactic_structure_options numerical_integration

function: genericradialmomentnumerical
Description: Returns the radial moment of the density in the dark matter profile of node between the given radiusMinimum and radiusMaximum (given in units of Mpc).
Code lines: 37
Contained by: module dark_matter_profiles_generic
Modules used: fgsl numerical_integration

function: integrandradialmoment
Description: Integrand for radial moment in a generic dark matter profile.
Code lines: 11
Contained by: function genericradialmomentnumerical

function: genericradialvelocitydispersionnumerical
Description: Returns the radial velocity dispersion (in km/s) in the dark matter profile of node at the given radius (given in units of Mpc).
Code lines: 37
Contained by: module dark_matter_profiles_generic
Modules used: fgsl numerical_constants_physical numerical_integration

function: `genericjeansequationintegrand`
Description: Integrand for generic drak matter profile Jeans equation.
Code lines: 11
Contained by: function `genericradialvelocitydispersionnumerical`

function: `genericradiusenclosingdensitynumerical`
Description: Returns the radius (in Mpc) in the dark matter profile of `node` which encloses the given `density` (given in units of $M_\odot/\mathrm{Mpc}^{-3}$).
Code lines: 19
Contained by: module `dark_matter_profiles_generic`
Modules used: `root_finder`

function: `genericradiusenclosingmassnumerical`
Description: Returns the radius (in Mpc) in the dark matter profile of `node` which encloses the given `mass` (given in units of $M_\odot$).
Code lines: 19
Contained by: module `dark_matter_profiles_generic`
Modules used: `root_finder`

function: `genericradiusfromspecificangularmomentumnumerical`
Description: Returns the radius (in Mpc) in `node` at which a circular orbit has the given `specificAngularMomentum` (given in units of km s^{-1} Mpc).
Code lines: 19
Contained by: module `dark_matter_profiles_generic`
Modules used: `root_finder`

function: `genericrotationnormalizationnumerical`
Description: Return the normalization of the rotation velocity vs. specific angular momentum relation.
Code lines: 11
Contained by: module `dark_matter_profiles_generic`
Modules used: `numerical_constants_math`

function: `integrandpotential`
Description: Integrand for gravitational potential in a generic dark matter profile.
Code lines: 12
Contained by: module `dark_matter_profiles_generic`
Modules used: `numerical_constants_physical`

function: `integrandtimefreefall`
Description: Integrand for freefall time in the halo.
Code lines: 15
Contained by: module `dark_matter_profiles_generic`
Modules used: `numerical_constants_astronomical`

function: `rootcircularvelocitymaximum`

Description: Root function used in finding the radius at which the maximum circular velocity occurs. Since for a spherical profile $V_{\rm c}^2(r) = {\rm G}M(r)/r$, then

$$\frac{{\rm d}V_{\rm c}^2}{{\rm d}r} = -\frac{{\rm G}M(r)}{r^2} + 4\pi{\rm G}\rho(r)r. \tag{18.3}$$

Therefore, the peak of the rotation curve satisfies $4\pi r^3\rho(r) - M(r) = 0$.

Code lines: 12

Contained by: module dark_matter_profiles_generic

Modules used: numerical_constants_math

function: rootdensity
Description: Root function used in finding the radius enclosing a given mean density.
Code lines: 8
Contained by: module dark_matter_profiles_generic
Modules used: numerical_constants_math

function: rootmass
Description: Root function used in finding the radius enclosing a given mass.
Code lines: 7
Contained by: module dark_matter_profiles_generic

function: rootradiusfreefall
Description: Root function used in finding the radius corresponding to a given freefall time.
Code lines: 13
Contained by: module dark_matter_profiles_generic
Modules used: fgsl numerical_integration

function: rootspecificangularmomentum
Description: Root function used in finding the radius enclosing a given specific angular momentum.
Code lines: 7
Contained by: module dark_matter_profiles_generic

file: dark_matter_profiles.heating.null.F90
Description: A null dark matter halo profile heating class.
Code lines: 90

interface: darkmatterprofileheatingnull
Description: Constructors for the null dark matter profile heating class.
Code lines: 3
Contained by: file dark_matter_profiles.heating.null.F90

function: nullconstructorparameters
Description: Constructor for the null dark matter profile heating scales class which takes a parameter set as input.
Code lines: 10
Contained by: file dark_matter_profiles.heating.null.F90
Modules used: input_parameters

function: nullspecificenergy
Description: Returns the specific energy of heating in the given node.
Code lines: 11
Contained by: file dark_matter_profiles.heating.null.F90

function: nullspecificenergygradient
Description: Returns the gradient of the specific energy of heating in the given node.
Code lines: 11
Contained by: file dark_matter_profiles.heating.null.F90

function: `nullspecificenergyiseverywherezero`
Description: Returns true if the specific energy is everywhere zero in the given `node`.
Code lines: 10
Contained by: file dark_matter_profiles.heating.null.F90

file: `dark_matter_profiles.heating.summation.F90`
Description: A dark matter halo profile heating class which sums over other heat sources.
Code lines: 193

interface: `darkmatterprofileheatingsummation`
Description: Constructors for the `summation` dark matter profile heating class.
Code lines: 4
Contained by: file dark_matter_profiles.heating.summation.F90

type: `heatsourcelist`
Code lines: 3
Contained by: file dark_matter_profiles.heating.summation.F90

function: `summationconstructorinternal`
Description: Internal constructor for the "summation" dark matter profile heating class.
Code lines: 14
Contained by: file dark_matter_profiles.heating.summation.F90

function: `summationconstructorparameters`
Description: Constructor for the `summation` dark matter profile heating class which takes a parameter set as input.
Code lines: 21
Contained by: file dark_matter_profiles.heating.summation.F90
Modules used: input_parameters

subroutine: `summationdeepcopy`
Description: Perform a deep copy for the `summation` dark matter halo profile heating class.
Code lines: 31
Contained by: file dark_matter_profiles.heating.summation.F90
Modules used: galacticus_error

subroutine: `summationdestructor`
Description: Destructor for the "summation" dark matter profile heating class.
Code lines: 14
Contained by: file dark_matter_profiles.heating.summation.F90

function: `summationspecificenergy`
Description: Returns the specific energy of heating in the given `node`.
Code lines: 16
Contained by: file dark_matter_profiles.heating.summation.F90

function: `summationspecificenergygradient`
Description: Returns the gradient of the specific energy of heating in the given `node`.

Code lines: 16
Contained by: file dark_matter_profiles.heating.summation.F90

function: summationspecificenergyiseverywherezero
Description: Returns true if the specific energy is everywhere zero in the given node.
Code lines: 18
Contained by: file dark_matter_profiles.heating.summation.F90

file: dark_matter_profiles.heating.tidal.F90
Description: A dark matter halo profile heating class which accounts for heating from tidal shocking.
Code lines: 103

interface: darkmatterprofileheatingtidal
Description: Constructors for the tidal dark matter profile heating class.
Code lines: 3
Contained by: file dark_matter_profiles.heating.tidal.F90

function: tidalconstructorparameters
Description: Constructor for the tidal dark matter profile heating scales class which takes a parameter set as input.
Code lines: 10
Contained by: file dark_matter_profiles.heating.tidal.F90
Modules used: input_parameters

function: tidalspecificenergy
Description: Returns the specific energy of heating in the given node.
Code lines: 16
Contained by: file dark_matter_profiles.heating.tidal.F90
Modules used: galacticus_nodes

function: tidalspecificenergygradient
Description: Returns the gradient of the specific energy of heating in the given node.
Code lines: 16
Contained by: file dark_matter_profiles.heating.tidal.F90
Modules used: galacticus_nodes

function: tidalspecificenergyiseverywherezero
Description: Returns true if the specific energy is everywhere zero in the given node.
Code lines: 13
Contained by: file dark_matter_profiles.heating.tidal.F90
Modules used: galacticus_nodes

file: dark_matter_profiles.heating.two_body_relaxation.F90
Description: A dark matter halo profile heating class which computes heating due to two-body relaxation.
Code lines: 170

interface: darkmatterprofileheatingtwobodyrelaxation
Description: Constructors for the twoBodyRelaxation dark matter profile heating class.
Code lines: 4

Contained by: file dark_matter_profiles.heating.two_body_relaxation.F90

function: twobodyrelaxationconstructorinternal
Description: Internal constructor for the twoBodyRelaxation dark matter profile heating scales class.
Code lines: 8
Contained by: file dark_matter_profiles.heating.two_body_relaxation.F90

function: twobodyrelaxationconstructorparameters
Description: Constructor for the twoBodyRelaxation dark matter profile heating scales class which takes a parameter set as input.
Code lines: 39
Contained by: file dark_matter_profiles.heating.two_body_relaxation.F90
Modules used: input_parameters

function: twobodyrelaxationspecificenergy
Description: Returns the specific energy of heating in the given node. The assumption here is that the mean fractional change in energy for a particle per crossing time is $8 \log \Lambda / N$ where N is the number of particles within radius $r =$radius. The crossing time is approximated by $r/V(r)$ where $V(r)$ is the circular velocity at r. The Coulomb logarithm is given by $\log \Lambda = \max(\epsilon, b_{90})$ where ϵ is the softening length, $b_{90} = 2 G m_\mathrm{p} / V^2(r)$, and m_p is the particle mass. Finally, the specific energy is assumed to be $\sigma^2(r)/2 \approx V^2(r)/4$.
Code lines: 30
Contained by: file dark_matter_profiles.heating.two_body_relaxation.F90
Modules used: galacticus_nodes numerical_constants_astronomical numerical_constants_physical numerical_constants_prefixes

function: twobodyrelaxationspecificenergygradient
Description: Returns the gradient of the specific energy of heating in the given node.
Code lines: 28
Contained by: file dark_matter_profiles.heating.two_body_relaxation.F90
Modules used: galacticus_nodes numerical_constants_physical

function: twobodyrelaxationspecificenergyiseverywherezero
Description: Returns true if the specific energy is everywhere zero in the given node.
Code lines: 13
Contained by: file dark_matter_profiles.heating.two_body_relaxation.F90
Modules used: galacticus_nodes

file: dark_matter_profiles.structure.concentration.Bullock2001.F90
Description: An implementation of dark matter halo profile concentrations using the Navarro et al. [1996] algorithm.
Code lines: 195
Modules used: cosmological_density_field cosmology_functions cosmology_parameters

function: bullock2001concentration
Description: Return the concentration of the dark matter halo profile of node using the Bullock et al. [2001] algorithm.
Code lines: 29
Contained by: file dark_matter_profiles.structure.concentration.Bullock2001.F90

Modules used: dark_matter_profile_mass_definitions galacticus_nodes virial_density_contrast

function: bullock2001constructorinternal
Description: Constructor for the bullock2001 dark matter halo profile concentration class.
Code lines: 22
Contained by: file dark_matter_profiles.structure.concentration.Bullock2001.F90
Modules used: dark_matter_halo_scales

function: bullock2001constructorparameters
Description: Default constructor for the bullock2001 dark matter halo profile concentration class.
Code lines: 43
Contained by: file dark_matter_profiles.structure.concentration.Bullock2001.F90
Modules used: input_parameters

function: bullock2001darkmatterprofiledefinition
Description: Return a dark matter density profile object defining that used in the definition of concentration in the Bullock et al. [2001] algorithm.
Code lines: 9
Contained by: file dark_matter_profiles.structure.concentration.Bullock2001.F90

function: bullock2001densitycontrastdefinition
Description: Return a virial density contrast object defining that used in the definition of concentration in the Bullock et al. [2001] algorithm.
Code lines: 9
Contained by: file dark_matter_profiles.structure.concentration.Bullock2001.F90

subroutine: bullock2001destructor
Description: Destructor for the bullock2001 dark matter halo profile concentration class.
Code lines: 12
Contained by: file dark_matter_profiles.structure.concentration.Bullock2001.F90

interface: darkmatterprofileconcentrationbullock2001
Description: Constructors for the bullock2001 dark matter halo profile concentration class.
Code lines: 4
Contained by: file dark_matter_profiles.structure.concentration.Bullock2001.F90

file: dark_matter_profiles.structure.concentration.Correa2015.F90
Description: An implementation of dark matter halo profile concentrations using the Correa et al. [2015] algorithm.
Code lines: 212
Modules used: cosmological_density_field cosmology_functions cosmology_parameters linear_growth

function: correa2015concentration
Description: Return the concentration of the dark matter halo profile of node using the Correa et al. [2015] algorithm.
Code lines: 57
Contained by: file dark_matter_profiles.structure.concentration.Correa2015.F90
Modules used: dark_matter_halos_correa2015 galacticus_nodes

root_finder

function: concentrationsolver
Description: Solver used in finding halo concentration using the Correa et al. [2015] algorithm.
Code lines: 19
Contained by: function correa2015concentration

function: correa2015constructorinternal
Description: Constructor for the correa2015 dark matter halo profile concentration class.
Code lines: 21
Contained by: file dark_matter_profiles.structure.concentration.Correa2015.F90
Modules used: dark_matter_halo_scales

function: correa2015constructorparameters
Description: Default constructor for the correa2015 dark matter halo profile concentration class.
Code lines: 32
Contained by: file dark_matter_profiles.structure.concentration.Correa2015.F90

function: correa2015darkmatterprofiledefinition
Description: Return a dark matter density profile object defining that used in the definition of concentration in the Correa et al. [2015] algorithm.
Code lines: 9
Contained by: file dark_matter_profiles.structure.concentration.Correa2015.F90

function: correa2015densitycontrastdefinition
Description: Return a virial density contrast object defining that used in the definition of concentration in the Correa et al. [2015] algorithm.
Code lines: 10
Contained by: file dark_matter_profiles.structure.concentration.Correa2015.F90

subroutine: correa2015destructor
Description: Destructor for the correa2015 dark matter profile concentration class.
Code lines: 12
Contained by: file dark_matter_profiles.structure.concentration.Correa2015.F90

interface: darkmatterprofileconcentrationcorrea2015
Description: Constructors for the correa2015 dark matter halo profile concentration class.
Code lines: 4
Contained by: file dark_matter_profiles.structure.concentration.Correa2015.F90

file: dark_matter_profiles.structure.concentration.Diemer-Kravtsov2014.F90
Description: An implementation of dark matter halo profile concentrations using the Diemer and Kravtsov [2014] algorithm.
Code lines: 274
Modules used: cosmological_density_field cosmology_functions cosmology_parameters power_spectra

interface: darkmatterprofileconcentrationdiemerkravtsov2014
Description: Constructors for the diemerKravtsov2014 dark matter halo profile concentration class.
Code lines: 4

Contained by: file dark_matter_profiles.structure.concentration.Diemer-Kravtsov2014.F90

function: diemerkravtsov2014concentration
Description: Return the concentration of the dark matter halo profile of node using the Diemer and Kravtsov [2014] algorithm.
Code lines: 12
Contained by: file dark_matter_profiles.structure.concentration.Diemer-Kravtsov2014.F90

function: diemerkravtsov2014concentrationmean
Description: Return the mean concentration of the dark matter halo profile of node using the Diemer and Kravtsov [2014] algorithm.
Code lines: 26
Contained by: file dark_matter_profiles.structure.concentration.Diemer-Kravtsov2014.F90
Modules used: galacticus_nodes math_exponentiation numerical_constants_math

function: diemerkravtsov2014constructorinternal
Description: Constructor for the diemerKravtsov2014 dark matter halo profile concentration class.
Code lines: 27
Contained by: file dark_matter_profiles.structure.concentration.Diemer-Kravtsov2014.F90
Modules used: dark_matter_halo_scales galacticus_error

function: diemerkravtsov2014constructorparameters
Description: Default constructor for the diemerKravtsov2014 dark matter halo profile concentration class.
Code lines: 100
Contained by: file dark_matter_profiles.structure.concentration.Diemer-Kravtsov2014.F90

function: diemerkravtsov2014darkmatterprofiledefinition
Description: Return a dark matter density profile object defining that used in the definition of concentration in the Diemer and Kravtsov [2014] algorithm.
Code lines: 10
Contained by: file dark_matter_profiles.structure.concentration.Diemer-Kravtsov2014.F90
Modules used: dark_matter_halo_scales

function: diemerkravtsov2014densitycontrastdefinition
Description: Return a virial density contrast object defining that used in the definition of concentration in the Diemer and Kravtsov [2014] algorithm.
Code lines: 9
Contained by: file dark_matter_profiles.structure.concentration.Diemer-Kravtsov2014.F90

subroutine: diemerkravtsov2014destructor
Description: Destructor for the diemerKravtsov2014 dark matter halo profile concentration class.
Code lines: 13
Contained by: file dark_matter_profiles.structure.concentration.Diemer-Kravtsov2014.F90

file: dark_matter_profiles.structure.concentration.Dutton-Maccio2014.F90
Description: An implementation of dark matter halo profile concentrations using the Dutton and Macciò [2014] algorithm.
Code lines: 311
Modules used: cosmology_functions cosmology_parameters

interface: `darkmatterprofileconcentrationduttonmaccio2014`
Description: Constructors for the `duttonMaccio2014` dark matter halo profile concentration class.
Code lines: 5
Contained by: file `dark_matter_profiles.structure.concentration.Dutton-Maccio2014.F90`

function: `duttonmaccio2014concentration`
Description: Return the concentration of the dark matter halo profile of `node` using the Dutton and Macciò [2014] algorithm.
Code lines: 21
Contained by: file `dark_matter_profiles.structure.concentration.Dutton-Maccio2014.F90`
Modules used: `galacticus_nodes`

function: `duttonmaccio2014constructorinternaldefined`
Description: Constructor for the `duttonMaccio2014` dark matter halo profile concentration class with user defined parameters.
Code lines: 13
Contained by: file `dark_matter_profiles.structure.concentration.Dutton-Maccio2014.F90`
Modules used: `galacticus_error`

function: `duttonmaccio2014constructorinternaltype`
Description: Constructor for the `duttonMaccio2014` dark matter halo profile concentration class.
Code lines: 43
Contained by: file `dark_matter_profiles.structure.concentration.Dutton-Maccio2014.F90`
Modules used: `galacticus_error`

function: `duttonmaccio2014constructorparameters`
Description: Default constructor for the `duttonMaccio2014` dark matter halo profile concentration class.
Code lines: 72
Contained by: file `dark_matter_profiles.structure.concentration.Dutton-Maccio2014.F90`

function: `duttonmaccio2014darkmatterprofiledefinition`
Description: Return a dark matter density profile object defining that used in the definition of concentration in the Diemer and Kravtsov [2014] algorithm.
Code lines: 9
Contained by: file `dark_matter_profiles.structure.concentration.Dutton-Maccio2014.F90`

subroutine: `duttonmaccio2014definitions`
Description: Establish virial density contrast and dark matter profile definitions.
Code lines: 39
Contained by: file `dark_matter_profiles.structure.concentration.Dutton-Maccio2014.F90`
Modules used: `dark_matter_halo_scales`

function: `duttonmaccio2014densitycontrastdefinition`
Description: Return a virial density contrast object defining that used in the definition of concentration in the Dutton and Macciò [2014] algorithm.
Code lines: 8
Contained by: file `dark_matter_profiles.structure.concentration.Dutton-Maccio2014.F90`

subroutine: `duttonmaccio2014destructor`

Description: Destructor for the DuttonMaccio2014 dark matter profile concentration class.
Code lines: 10
Contained by: file dark_matter_profiles.structure.concentration.Dutton-Maccio2014.F90

file: dark_matter_profiles.structure.concentration.F90
Description: Contains a module which provides an object that implements concentrations of dark matter halo profiles.
Code lines: 61

module: dark_matter_profiles_concentration
Description: Provides a class that implements concentrations of dark matter halo profiles.
Code lines: 39
Contained by: file dark_matter_profiles.structure.concentration.F90
Modules used: dark_matter_profiles_dmo, galacticus_nodes, virial_density_contrast
Used by: file dark_matter_profiles.structure.scale.concentration.F90; subroutine galacticus_function_classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; file merger_trees.construct.read.F90; module virial_density_contrast_percolation_utilities; program test_diemerkravtsov2014_concentration; program test_nfw96_concentration_dark_energy; program test_prada2011_concentration; program test_zhao2009_flat; program test_zhao2009_dark_energy; program test_zhao2009_open; program test_correa2015_concentration

file: dark_matter_profiles.structure.concentration.Gao2008.F90
Description: An implementation of dark matter halo profile concentrations using the Gao et al. [2008] algorithm.
Code lines: 150
Modules used: cosmology_functions, cosmology_parameters

interface: darkmatterprofileconcentrationgao2008
Description: Constructors for the gao2008 dark matter halo profile concentration class.
Code lines: 4
Contained by: file dark_matter_profiles.structure.concentration.Gao2008.F90

function: gao2008concentration
Description: Return the concentration of the dark matter halo profile of node using the Gao et al. [2008] algorithm.
Code lines: 20
Contained by: file dark_matter_profiles.structure.concentration.Gao2008.F90
Modules used: galacticus_nodes

function: gao2008constructorinternal
Description: Internal constructor for the gao2008 dark matter halo profile concentration class.
Code lines: 18
Contained by: file dark_matter_profiles.structure.concentration.Gao2008.F90
Modules used: dark_matter_halo_scales

function: gao2008constructorparameters
Description: Constructor for the gao2008 dark matter halo profile concentration class which takes a parameter list as input.
Code lines: 17
Contained by: file dark_matter_profiles.structure.concentration.Gao2008.F90
Modules used: input_parameters

function: gao2008darkmatterprofiledefinition
Description: Return a dark matter density profile object defining that used in the definition of concentration in the Gao et al. [2008] algorithm.
Code lines: 9
Contained by: file dark_matter_profiles.structure.concentration.Gao2008.F90

function: gao2008densitycontrastdefinition
Description: Return a virial density contrast object defining that used in the definition of concentration in the Gao et al. [2008] algorithm.
Code lines: 9
Contained by: file dark_matter_profiles.structure.concentration.Gao2008.F90

subroutine: gao2008destructor
Description: Destructor for the gao2008 dark matter halo profile concentration class.
Code lines: 10
Contained by: file dark_matter_profiles.structure.concentration.Gao2008.F90

file: dark_matter_profiles.structure.concentration.Klypin2015.F90
Description: An implementation of dark matter halo profile concentrations using the Klypin et al. [2014] algorithm.
Code lines: 372
Modules used: cosmological_density_field cosmology_functions cosmology_parameters tables

interface: darkmatterprofileconcentrationklypin2015
Description: Constructors for the klypin2015 dark matter halo profile concentration class.
Code lines: 4
Contained by: file dark_matter_profiles.structure.concentration.Klypin2015.F90

function: klypin2015concentration
Description: Return the concentration of the dark matter halo profile of node using the Klypin et al. [2014] algorithm.
Code lines: 39
Contained by: file dark_matter_profiles.structure.concentration.Klypin2015.F90
Modules used: galacticus_error galacticus_nodes

function: klypin2015constructorinternal
Description: Constructor for the klypin2015 dark matter halo profile concentration class.
Code lines: 158
Contained by: file dark_matter_profiles.structure.concentration.Klypin2015.F90
Modules used: dark_matter_halo_scales table_labels

function: `klypin2015constructorparameters`
Description: Default constructor for the `klypin2015` dark matter halo profile concentration class.
Code lines: 30
Contained by: file `dark_matter_profiles.structure.concentration.Klypin2015.F90`
Modules used: `galacticus_error`

function: `klypin2015darkmatterprofiledefinition`
Description: Return a dark matter density profile object defining that used in the definition of concentration in the Klypin et al. [2014] algorithm.
Code lines: 9
Contained by: file `dark_matter_profiles.structure.concentration.Klypin2015.F90`

function: `klypin2015densitycontrastdefinition`
Description: Return a virial density contrast object defining that used in the definition of concentration in the Klypin et al. [2014] algorithm.
Code lines: 9
Contained by: file `dark_matter_profiles.structure.concentration.Klypin2015.F90`

subroutine: `klypin2015destructor`
Description: Destructor for the `klypin2015` dark matter halo profile concentration class.
Code lines: 12
Contained by: file `dark_matter_profiles.structure.concentration.Klypin2015.F90`

file: `dark_matter_profiles.structure.concentration.Ludlow2016_fit.F90`
Description: An implementation of dark matter halo profile concentrations using the Ludlow et al. [2016] fitting function.
Code lines: 160
Modules used: `cosmological_density_field` `cosmology_functions` `cosmology_parameters`

interface: `darkmatterprofileconcentrationludlow2016fit`
Description: Constructors for the `ludlow2016Fit` dark matter halo profile concentration class.
Code lines: 4
Contained by: file `dark_matter_profiles.structure.concentration.Ludlow2016_fit.F90`

function: `ludlow2016fitconcentration`
Description: Return the concentration of the dark matter halo profile of `node` using the Ludlow et al. [2016] fitting function.
Code lines: 23
Contained by: file `dark_matter_profiles.structure.concentration.Ludlow2016_fit.F90`
Modules used: `galacticus_error` `galacticus_nodes`

function: `ludlow2016fitconstructorinternal`
Description: Constructor for the `ludlow2016Fit` dark matter halo profile concentration class.
Code lines: 19
Contained by: file `dark_matter_profiles.structure.concentration.Ludlow2016_fit.F90`
Modules used: `dark_matter_halo_scales`

function: `ludlow2016fitconstructorparameters`
Description: Default constructor for the `ludlow2016Fit` dark matter halo profile concentration class.
Code lines: 19
Contained by: file `dark_matter_profiles.structure.concentration.Ludlow2016_fit.F90`
Modules used: `input_parameters`

function: `ludlow2016fitdarkmatterprofiledefinition`
Description: Return a dark matter density profile object defining that used in the definition of concentration in the Ludlow et al. [2016] algorithm.
Code lines: 9
Contained by: file `dark_matter_profiles.structure.concentration.Ludlow2016_fit.F90`

function: `ludlow2016fitdensitycontrastdefinition`
Description: Return a virial density contrast object defining that used in the definition of concentration in the Diemer and Kravtsov [2014] algorithm.
Code lines: 9
Contained by: file `dark_matter_profiles.structure.concentration.Ludlow2016_fit.F90`

subroutine: `ludlow2016fitdestructor`
Description: Destructor for the `ludlow2016Fit` dark matter halo profile concentration class.
Code lines: 11
Contained by: file `dark_matter_profiles.structure.concentration.Ludlow2016_fit.F90`

file: `dark_matter_profiles.structure.concentration.MunozCuartas2011.F90`
Description: An implementation of dark matter halo profile concentrations using the Muñoz-Cuartas et al. [2011] algorithm.
Code lines: 150
Modules used: `cosmology_functions` `cosmology_parameters`

interface: `darkmatterprofileconcentrationmunozcuartas2011`
Description: Constructors for the `munozCuartas2011` dark matter halo profile concentration class.
Code lines: 4
Contained by: file `dark_matter_profiles.structure.concentration.MunozCuartas2011.F90`

function: `munozcuartas2011concentration`
Description: Return the concentration of the dark matter halo profile of `node` using the Muñoz-Cuartas et al. [2011] algorithm.
Code lines: 22
Contained by: file `dark_matter_profiles.structure.concentration.MunozCuartas2011.F90`
Modules used: `galacticus_nodes`

function: `munozcuartas2011constructorinternal`
Description: Internal constructor for the `munozCuartas2011` dark matter halo profile concentration class.
Code lines: 18
Contained by: file `dark_matter_profiles.structure.concentration.MunozCuartas2011.F90`
Modules used: `dark_matter_halo_scales`

function: `munozcuartas2011constructorparameters`

Description: Constructor for the `munozCuartas2011` dark matter halo profile concentration class which takes a parameter set as input.
Code lines: 16
Contained by: file `dark_matter_profiles.structure.concentration.MunozCuartas2011.F90`
Modules used: `input_parameters`

function: `munozcuartas2011darkmatterprofiledefinition`
Description: Return a dark matter density profile object defining that used in the definition of concentration in the Muñoz-Cuartas et al. [2011] algorithm.
Code lines: 9
Contained by: file `dark_matter_profiles.structure.concentration.MunozCuartas2011.F90`

function: `munozcuartas2011densitycontrastdefinition`
Description: Return a virial density contrast object defining that used in the definition of concentration in the Muñoz-Cuartas et al. [2011] algorithm.
Code lines: 9
Contained by: file `dark_matter_profiles.structure.concentration.MunozCuartas2011.F90`

subroutine: `munozcuartas2011destructor`
Description: Destructor for the `munozCuartas2011` dark matter halo profile concentration class.
Code lines: 10
Contained by: file `dark_matter_profiles.structure.concentration.MunozCuartas2011.F90`

file: `dark_matter_profiles.structure.concentration.NFW.F90`
Description: An implementation of dark matter halo profile concentrations using the Navarro et al. [1996] algorithm.
Code lines: 229
Modules used: `cosmological_density_field` `cosmology_functions` `cosmology_parameters` `virial_density_contrast`

interface: `darkmatterprofileconcentrationnfw1996`
Description: Constructors for the `nfw1996` dark matter halo profile concentration class.
Code lines: 4
Contained by: file `dark_matter_profiles.structure.concentration.NFW.F90`

function: `nfw1996concentration`
Description: Return the concentration of the dark matter halo profile of `node` using the Navarro et al. [1996] algorithm.
Code lines: 41
Contained by: file `dark_matter_profiles.structure.concentration.NFW.F90`
Modules used: `galacticus_nodes` `root_finder` `virial_density_contrast`

function: `nfw1996concentrationroot`
Description: Root function used in finding concentrations in the Navarro et al. [1996] method.
Code lines: 7
Contained by: file `dark_matter_profiles.structure.concentration.NFW.F90`

function: `nfw1996constructorinternal`
Description: Constructor for the `nfw1996` dark matter halo profile concentration class.

Code lines: 23
Contained by: file dark_matter_profiles.structure.concentration.NFW.F90
Modules used: dark_matter_halo_scales

function: nfw1996constructorparameters
Description: Default constructor for the nfw1996 dark matter halo profile concentration class.
Code lines: 48
Contained by: file dark_matter_profiles.structure.concentration.NFW.F90
Modules used: input_parameters

function: nfw1996darkmatterprofiledefinition
Description: Return a dark matter density profile object defining that used in the definition of concentration in the Navarro et al. [1996] algorithm.
Code lines: 9
Contained by: file dark_matter_profiles.structure.concentration.NFW.F90

function: nfw1996densitycontrastdefinition
Description: Return a virial density contrast object defining that used in the definition of concentration in the Navarro et al. [1996] algorithm.
Code lines: 9
Contained by: file dark_matter_profiles.structure.concentration.NFW.F90

subroutine: nfw1996destructor
Description: Destructor for the nfw1996 dark matter halo profile concentration class.
Code lines: 13
Contained by: file dark_matter_profiles.structure.concentration.NFW.F90

file: dark_matter_profiles.structure.concentration.Prada2011.F90
Description: An implementation of dark matter halo profile concentrations using the Prada et al. [2011] algorithm.
Code lines: 330
Modules used: cosmological_density_field, cosmology_functions, cosmology_parameters, linear_growth

interface: darkmatterprofileconcentrationprada2011
Description: Constructors for the prada2011 dark matter halo profile concentration class.
Code lines: 4
Contained by: file dark_matter_profiles.structure.concentration.Prada2011.F90

function: prada2011b0
Description: The function $B_0(x)$ as defined in eqn. (18) of Prada et al. [2011].
Code lines: 7
Contained by: file dark_matter_profiles.structure.concentration.Prada2011.F90

function: prada2011b1
Description: The function $B_1(x)$ as defined in eqn. (18) of Prada et al. [2011].
Code lines: 7
Contained by: file dark_matter_profiles.structure.concentration.Prada2011.F90

function: `prada2011c`
Description: The function $\mathcal{C}(\sigma')$ as defined in eqn. (17) of Prada et al. [2011].
Code lines: 7
Contained by: file `dark_matter_profiles.structure.concentration.Prada2011.F90`

function: `prada2011cmin`
Description: The function $c_{\rm min}(x)$ as defined in eqn. (19) of Prada et al. [2011].
Code lines: 8
Contained by: file `dark_matter_profiles.structure.concentration.Prada2011.F90`
Modules used: `numerical_constants_math`

function: `prada2011concentration`
Description: Return the concentration of the dark matter halo profile of `node` using the Prada et al. [2011] algorithm.
Code lines: 17
Contained by: file `dark_matter_profiles.structure.concentration.Prada2011.F90`
Modules used: `galacticus_nodes`

function: `prada2011constructorinternal`
Description: Constructor for the `prada2011` dark matter halo profile concentration class.
Code lines: 21
Contained by: file `dark_matter_profiles.structure.concentration.Prada2011.F90`
Modules used: `dark_matter_halo_scales`

function: `prada2011constructorparameters`
Description: Default constructor for the `prada2011` dark matter halo profile concentration class.
Code lines: 145
Contained by: file `dark_matter_profiles.structure.concentration.Prada2011.F90`
Modules used: `input_parameters`

function: `prada2011darkmatterprofiledefinition`
Description: Return a dark matter density profile object defining that used in the definition of concentration in the Prada et al. [2011] algorithm.
Code lines: 9
Contained by: file `dark_matter_profiles.structure.concentration.Prada2011.F90`

function: `prada2011densitycontrastdefinition`
Description: Return a virial density contrast object defining that used in the definition of concentration in the Prada et al. [2011] algorithm.
Code lines: 8
Contained by: file `dark_matter_profiles.structure.concentration.Prada2011.F90`

subroutine: `prada2011destructor`
Description: Destructor for the `prada2011` dark matter halo profile concentration class.
Code lines: 12
Contained by: file `dark_matter_profiles.structure.concentration.Prada2011.F90`

function: `prada2011inversesigmamin`

Description: The function $\sigma_{\rm min}^{-1}(x)$ as defined in eqn. (20) of Prada et al. [2011].
Code lines: 8
Contained by: file dark_matter_profiles.structure.concentration.Prada2011.F90
Modules used: numerical_constants_math

file: dark_matter_profiles.structure.concentration.Schneider2015.F90
Description: An implementation of halo profile concentrations using the algorithm of Schneider [2015].
Code lines: 202
Modules used: cosmological_density_field cosmology_functions root_finder

interface: darkmatterprofileconcentrationschneider2015
Description: Constructors for the Schneider2015 dark matter halo profile concentration class.
Code lines: 4
Contained by: file dark_matter_profiles.structure.concentration.Schneider2015.F90

function: schneider2015concentration
Description: Return the concentration of the dark matter halo profile of node using the algorithm of Schneider [2015].
Code lines: 43
Contained by: file dark_matter_profiles.structure.concentration.Schneider2015.F90
Modules used: galacticus_nodes numerical_constants_math

function: schneider2015constructorinternal
Description: Generic constructor for the schneider2015 dark matter halo concentration class.
Code lines: 11
Contained by: file dark_matter_profiles.structure.concentration.Schneider2015.F90

function: schneider2015constructorparameters
Description: Default constructor for the schneider2015 dark matter halo profile concentration class.
Code lines: 31
Contained by: file dark_matter_profiles.structure.concentration.Schneider2015.F90
Modules used: galacticus_error input_parameters

function: schneider2015darkmatterprofiledefinition
Description: Return a dark matter density profile object defining that used in the definition of concentration in the Schneider [2015] algorithm.
Code lines: 10
Contained by: file dark_matter_profiles.structure.concentration.Schneider2015.F90
Modules used: dark_matter_halo_scales

function: schneider2015densitycontrastdefinition
Description: Return a virial density contrast object defining that used in the definition of concentration in the Schneider [2015] algorithm.
Code lines: 9
Contained by: file dark_matter_profiles.structure.concentration.Schneider2015.F90

subroutine: schneider2015destructor
Description: Destructor for the schneider2015 dark matter halo profile concentration class.

Code lines: 13
Contained by: file dark_matter_profiles.structure.concentration.Schneider2015.F90

function: schneider2015referencecollapsemassroot
Description: Root function used to find the mass collapsing at given time in dark matter halo concentration algorithm of Schneider [2015].
Code lines: 15
Contained by: file dark_matter_profiles.structure.concentration.Schneider2015.F90
Modules used: numerical_constants_math

file: dark_matter_profiles.structure.concentration.WDM.F90
Description: An implementation of warm dark matter halo profile concentrations using the Schneider et al. [2012] modifier.
Code lines: 129
Modules used: transfer_functions

interface: darkmatterprofileconcentrationwdm
Description: Constructors for the WDM dark matter halo profile concentration class.
Code lines: 5
Contained by: file dark_matter_profiles.structure.concentration.WDM.F90

function: wdmconcentration
Description: Return the concentration of the dark matter halo profile of node using the warm dark matter modifier of Schneider et al. [2012].
Code lines: 16
Contained by: file dark_matter_profiles.structure.concentration.WDM.F90
Modules used: galacticus_nodes

function: wdmconstructorinternal
Description: Generic constructor for the WDM dark matter halo concentration class.
Code lines: 9
Contained by: file dark_matter_profiles.structure.concentration.WDM.F90

function: wdmconstructorparameters
Description: Default constructor for the wdm dark matter halo profile concentration class.
Code lines: 16
Contained by: file dark_matter_profiles.structure.concentration.WDM.F90
Modules used: input_parameters

function: wdmdarkmatterprofiledefinition
Description: Return a dark matter density profile object defining that used in the definition of concentration in the warm dark matter modifier of Schneider et al. [2012].
Code lines: 9
Contained by: file dark_matter_profiles.structure.concentration.WDM.F90

function: wdmdensitycontrastdefinition
Description: Return a virial density contrast object defining that used in the definition of concentration in the warm dark matter modifier of Schneider et al. [2012].
Code lines: 9
Contained by: file dark_matter_profiles.structure.concentration.WDM.F90

subroutine: `wdmdestructor`
Description: Destructor for the `wdm` dark matter halo profile concentration class.
Code lines: 8
Contained by: file `dark_matter_profiles.structure.concentration.WDM.F90`

file: `dark_matter_profiles.structure.concentration.Zhao2009.F90`
Description: An implementation of dark matter halo profile concentrations using the Zhao et al. [2009] algorithm.
Code lines: 163
Modules used: `cosmology_functions` `cosmology_parameters` `dark_matter_halo_mass_accretion_histories`

interface: `darkmatterprofileconcentrationzhao2009`
Description: Constructors for the `zhao2009` dark matter halo profile concentration class.
Code lines: 5
Contained by: file `dark_matter_profiles.structure.concentration.Zhao2009.F90`

function: `zhao2009concentration`
Description: Return the concentration of the dark matter halo profile of `node` using the Zhao et al. [2009] algorithm.
Code lines: 22
Contained by: file `dark_matter_profiles.structure.concentration.Zhao2009.F90`
Modules used: `dark_matter_halo_formation_times` `galacticus_nodes`

function: `zhao2009constructorinternal`
Description: Internal constructor for the `zhao2009` dark matter halo profile concentration class.
Code lines: 20
Contained by: file `dark_matter_profiles.structure.concentration.Zhao2009.F90`
Modules used: `dark_matter_halo_scales`

function: `zhao2009constructorparameters`
Description: Constructor for the `zhao2009` dark matter halo profile concentration class which takes an input parameter list.
Code lines: 20
Contained by: file `dark_matter_profiles.structure.concentration.Zhao2009.F90`
Modules used: `input_parameters`

function: `zhao2009darkmatterprofiledefinition`
Description: Return a dark matter density profile object defining that used in the definition of concentration in the Zhao et al. [2009] algorithm.
Code lines: 9
Contained by: file `dark_matter_profiles.structure.concentration.Zhao2009.F90`

function: `zhao2009densitycontrastdefinition`
Description: Return a virial density contrast object defining that used in the definition of concentration in the Zhao et al. [2009] algorithm.
Code lines: 9
Contained by: file `dark_matter_profiles.structure.concentration.Zhao2009.F90`

subroutine: zhao2009destructor
Description: Destructor for the zhao2009 dark matter halo profile concentration class.
Code lines: 11
Contained by: file dark_matter_profiles.structure.concentration.Zhao2009.F90

file: dark_matter_profiles.structure.mass_definitions.F90
Description: Contains a module which implements conversions of total halo mass between different definitions.
Code lines: 79

module: dark_matter_profile_mass_definitions
Description: Implements calculations of dark matter profile scale radii from concentrations.
Code lines: 57
Contained by: file dark_matter_profiles.structure.mass_definitions.F90
Used by:

function naozbarkana2007filteredfraction	function naozbarkana2007filteredfractionrate
function bullock2001concentration	function ludlow2014formationtimeroot
function ludlow2016formationtimeroot	function ludlow2016radius
function halomasspasses	function concentrationextract
function masshaloextract	subroutine node_component_basic_-extended_bertschinger_solver
subroutine node_component_satellite_-orbiting_bound_mass_initialize	function benson2005orbit
function benson2005velocitytangentialmagnitudemean	function benson2005velocitytotalrootmeansquared
function jiang2014orbit	function jiang2014velocitytangentialmagnitudemean
function jiang2014velocitytotalrootmeansquared	function wetzel2010orbit
function fixedorbit	function fixedvelocitytangentialmagnitudemean
function fixedvelocitytotalrootmeansquared	

function: dark_matter_profile_mass_definition
Description: Compute the mass of node under the given density contrast definition.
Code lines: 48
Contained by: module dark_matter_profile_mass_definitions
Modules used:

cosmology_functions	cosmology_parameters
dark_matter_profiles_dmo	galacticus_nodes
math_exponentiation	numerical_comparison
numerical_constants_math	numerical_constants_physical
virial_density_contrast	

file: dark_matter_profiles.structure.scale.F90
Description: Contains a module which provides an object that implements concentrations of dark matter halo profiles.
Code lines: 42

module: `dark_matter_profile_scales`

Description: Provides a class that implements scale radii dark matter halo profiles.
Code lines: 20
Contained by: file dark_matter_profiles.structure.scale.F90
Modules used: galacticus_nodes
Used by:
- file dark_matter_halos.spins.distributions.N-body_errors.F90
- file galacticus.output.analyses.spin_distribution.Bett2007.F90
- subroutine galacticus_state_store
- file merger_trees.construct.read.F90
- file models.likelihoods.spin_distribution.F90
- module node_component_spin_vitvitska
- module virial_density_contrast_percolation_utilities
- subroutine halomassfunctionperform
- file tasks.halo_model_generate.F90
- subroutine galacticus_function_classes_destroy
- subroutine galacticus_state_retrieve
- subroutine halo_model_projected_correlation
- file models.likelihoods.projected_correlation_function.F90
- module node_component_dark_matter_profile_scale
- subroutine node_component_spin_vitvitska_thread_initialize
- file tasks.halo_mass_function.F90
- file tasks.halo_model.projected_correlation_function.F90

file: `dark_matter_profiles.structure.scale.Ludlow2014.F90`

Description: An implementation of dark matter halo profile concentrations using the Ludlow et al. [2014] algorithm.
Code lines: 141

interface: `darkmatterprofilescaleradiusludlow2014`

Description: Constructors for the `ludlow2014` dark matter halo profile concentration class.
Code lines: 4
Contained by: file dark_matter_profiles.structure.scale.Ludlow2014.F90

function: `ludlow2014constructorinternal`

Description: Constructor for the `ludlow2014` dark matter halo profile concentration class.
Code lines: 13
Contained by: file dark_matter_profiles.structure.scale.Ludlow2014.F90
Modules used: galacticus_error

function: `ludlow2014constructorparameters`

Description: Default constructor for the `ludlow2014` dark matter halo profile concentration class.
Code lines: 9
Contained by: file dark_matter_profiles.structure.scale.Ludlow2014.F90
Modules used: input_parameters

function: `ludlow2014formationtimeroot`

Description: Function used to find the formation time of a halo in the `ludlow2014` concentration algorithm.

Code lines: 41
Contained by: file dark_matter_profiles.structure.scale.Ludlow2014.F90
Modules used: dark_matter_profile_mass_definitions galacticus_nodes

subroutine: ludlow2014formationtimerootfunctionset
Description: Initialize the finder object to compute the relevant formation history.
Code lines: 14
Contained by: file dark_matter_profiles.structure.scale.Ludlow2014.F90
Modules used: root_finder

file: dark_matter_profiles.structure.scale.Ludlow2016.F90
Description: An implementation of dark matter halo profile concentrations using the Ludlow et al. [2016] algorithm.
Code lines: 399
Modules used: cosmology_functions cosmology_parameters
dark_matter_profiles_dmo root_finder

interface: darkmatterprofilescaleradiusludlow2016
Description: Constructors for the ludlow2016 dark matter halo profile concentration class.
Code lines: 4
Contained by: file dark_matter_profiles.structure.scale.Ludlow2016.F90

function: ludlow2016constructorinternal
Description: Constructor for the ludlow2016 dark matter halo profile concentration class.
Code lines: 15
Contained by: file dark_matter_profiles.structure.scale.Ludlow2016.F90
Modules used: galacticus_error

function: ludlow2016constructorparameters
Description: Default constructor for the ludlow2016 dark matter halo profile concentration class.
Code lines: 49
Contained by: file dark_matter_profiles.structure.scale.Ludlow2016.F90
Modules used: galacticus_error input_parameters

function: ludlow2016densitycontrast
Description: Compute the density contrast for this epoch.
Code lines: 12
Contained by: file dark_matter_profiles.structure.scale.Ludlow2016.F90

subroutine: ludlow2016destructor
Description: Destructor for the ludlow2016 dark matter halo profile concentration class.
Code lines: 10
Contained by: file dark_matter_profiles.structure.scale.Ludlow2016.F90

function: ludlow2016formationtimeroot
Description: Function used to find the formation time of a halo in the ludlow2016 concentration algorithm.
Code lines: 49
Contained by: file dark_matter_profiles.structure.scale.Ludlow2016.F90
Modules used: dark_matter_profile_mass_definitions galacticus_nodes

merger_tree_walkers

subroutine: ludlow2016formationtimerootfunctionset
Description: Initialize the finder object to compute the relevant formation history.
Code lines: 14
Contained by: file dark_matter_profiles.structure.scale.Ludlow2016.F90
Modules used: root_finder

function: ludlow2016radius
Description: Return the concentration of the dark matter halo profile of node using the Ludlow et al. [2016] algorithm.
Code lines: 148
Contained by: file dark_matter_profiles.structure.scale.Ludlow2016.F90
Modules used: dark_matter_profile_mass_definitions, galacticus_calculations_resets, galacticus_display, galacticus_error, galacticus_nodes, merger_tree_walkers, numerical_comparison, numerical_constants_math

type: ludlow2016state
Code lines: 6
Contained by: file dark_matter_profiles.structure.scale.Ludlow2016.F90

file: dark_matter_profiles.structure.scale.binary.F90
Description: An implementation of dark matter halo profile scale radii which switches between two methods based on a filter.
Code lines: 103
Modules used: galactic_filters

function: binaryconstructorinternal
Description: Constructor for the binary dark matter halo profile concentration class.
Code lines: 10
Contained by: file dark_matter_profiles.structure.scale.binary.F90
Modules used: galacticus_error

function: binaryconstructorparameters
Description: Default constructor for the binary dark matter halo profile concentration class.
Code lines: 21
Contained by: file dark_matter_profiles.structure.scale.binary.F90
Modules used: galacticus_error, input_parameters

subroutine: binarydestructor
Description: Destructor for the binary dark matter halo profile concentration class.
Code lines: 9
Contained by: file dark_matter_profiles.structure.scale.binary.F90

function: binaryradius
Description: Return the scale radius of the dark matter halo profile of node.
Code lines: 12
Contained by: file dark_matter_profiles.structure.scale.binary.F90

interface: `darkmatterprofilescaleradiusbinary`
Description: Constructors for the `binary` dark matter halo profile concentration class.
Code lines: 4
Contained by: file dark_matter_profiles.structure.scale.binary.F90

file: `dark_matter_profiles.structure.scale.concentration.F90`
Description: An implementation of dark matter halo profile scale radii in which radii are computed from the concentration.
Code lines: 307
Modules used: cosmology_functions, cosmology_parameters, dark_matter_halo_scales, dark_matter_profiles_concentration, dark_matter_profiles_dmo, galacticus_nodes, virial_density_contrast

function: `concentrationconstructorinternal`
Description: Internal constructor for the `concentration` dark matter halo profile scale radius class.
Code lines: 20
Contained by: file dark_matter_profiles.structure.scale.concentration.F90

function: `concentrationconstructorparameters`
Description: Constructor for the `concentration` dark matter halo profile scale radius class which takes a parameter list as input.
Code lines: 48
Contained by: file dark_matter_profiles.structure.scale.concentration.F90
Modules used: input_parameters

subroutine: `concentrationdestructor`
Description: Destructor for the `concentration` dark matter halo profile scale radius class.
Code lines: 15
Contained by: file dark_matter_profiles.structure.scale.concentration.F90

function: `concentrationmassroot`
Description: Root function used to find the mass of a halo corresponding to the definition used for a particular concentration class.
Code lines: 35
Contained by: file dark_matter_profiles.structure.scale.concentration.F90
Modules used: galacticus_calculations_resets

function: `concentrationradius`
Description: Compute the scale radius of the dark matter profile of `node`.
Code lines: 105
Contained by: file dark_matter_profiles.structure.scale.concentration.F90
Modules used: galacticus_calculations_resets, numerical_comparison, numerical_constants_math, root_finder

type: `concentrationstate`
Code lines: 6
Contained by: file dark_matter_profiles.structure.scale.concentration.F90

interface: `darkmatterprofilescaleradiusconcentration`
Description: Constructors for the `concentration` dark matter halo profile scale radius class.
Code lines: 4
Contained by: file dark_matter_profiles.structure.scale.concentration.F90

file: `dark_matter_profiles.structure.scale.zero.F90`
Description: An implementation of dark matter halo profile scale radii which returns zero radii—useful when scale radii are not relevant.
Code lines: 62

interface: `darkmatterprofilescaleradiuszero`
Description: Constructors for the `zero` dark matter halo profile scale radius class.
Code lines: 3
Contained by: file dark_matter_profiles.structure.scale.zero.F90

function: `zeroconstructorparameters`
Description: Constructor for the `zero` dark matter halo profile scale radius class which takes a parameter list as input.
Code lines: 11
Contained by: file dark_matter_profiles.structure.scale.zero.F90
Modules used: input_parameters

function: `zeroradius`
Description: Compute the scale radius of the dark matter profile of `node`.
Code lines: 9
Contained by: file dark_matter_profiles.structure.scale.zero.F90

file: `dark_matter_profiles.structure.shape.F90`
Description: Contains a module which provides a class that implements concentrations of dark matter halo profiles.
Code lines: 40

module: `dark_matter_profiles_shape`
Description: Provides a class that implements shape parameters of dark matter halo profiles.
Code lines: 18
Contained by: file dark_matter_profiles.structure.shape.F90
Modules used: galacticus_nodes
Used by: subroutine galacticus_function_classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; module node_component_dark_matter_profile_scale_shape; module virial_density_contrast_percolation_utilities

file: `dark_matter_profiles.structure.shape.Gao2008.F90`
Description: An implementation of dark matter halo profile shapes using the Gao et al. [2008] algorithm.
Code lines: 111
Modules used: cosmological_density_field

interface: darkmatterprofileshapegao2008
Description: Constructors for the gao2008 dark matter halo profile shape class.
Code lines: 4
Contained by: file dark_matter_profiles.structure.shape.Gao2008.F90

function: gao2008constructorinternal
Description: Constructor for the gao2008 dark matter halo profile shape class.
Code lines: 10
Contained by: file dark_matter_profiles.structure.shape.Gao2008.F90

function: gao2008constructorparameters
Description: Default constructor for the gao2008 dark matter halo profile shape class.
Code lines: 17
Contained by: file dark_matter_profiles.structure.shape.Gao2008.F90
Modules used: input_parameters

subroutine: gao2008destructor
Description: Destructor for the gao2008 dark matter halo profile shape class.
Code lines: 8
Contained by: file dark_matter_profiles.structure.shape.Gao2008.F90

function: gao2008shape
Description: Return the Einasto profile shape parameter of the dark matter halo profile of node using the Gao et al. [2008] algorithm. More specifically, the parameter is given by:

$$\alpha = \begin{cases} 0.155 + 0.0095\nu^2 & \text{if } \nu < 3.907 \\ 0.3 & \text{if } \nu \geq 3.907, \end{cases} \tag{18.4}$$

where $\nu = \delta_c(t)/\sigma(M)$ is the peak height of the halo.
Code lines: 25
Contained by: file dark_matter_profiles.structure.shape.Gao2008.F90
Modules used: galacticus_nodes

file: dark_matter_profiles.structure.shape.Klypin2015.F90
Description: An implementation of dark matter halo profile shapes using the Klypin et al. [2014] algorithm.
Code lines: 135
Modules used: cosmological_density_field

interface: darkmatterprofileshapeklypin2015
Description: Constructors for the klypin2015 dark matter halo profile shape parameter class.
Code lines: 4
Contained by: file dark_matter_profiles.structure.shape.Klypin2015.F90

function: klypin2015constructorinternal
Description: Constructor for the klypin2015 dark matter halo profile shape class.
Code lines: 13
Contained by: file dark_matter_profiles.structure.shape.Klypin2015.F90

Modules used: galacticus_error

function: klypin2015constructorparameters
Description: Default constructor for the klypin2015 dark matter halo profile shape class.
Code lines: 27
Contained by: file dark_matter_profiles.structure.shape.Klypin2015.F90
Modules used: input_parameters

subroutine: klypin2015destructor
Description: Destructor for the klypin2015 dark matter halo profile shape class.
Code lines: 8
Contained by: file dark_matter_profiles.structure.shape.Klypin2015.F90

function: klypin2015shape
Description: Return the Einasto profile shape parameter of the dark matter halo profile of node using the Klypin et al. [2014] algorithm.
Code lines: 25
Contained by: file dark_matter_profiles.structure.shape.Klypin2015.F90
Modules used: galacticus_error galacticus_nodes

file: dark_matter_profiles.structure_tasks.F90
Description: Contains a module which implements structure tasks related to the dark matter halo density profile.
Code lines: 172

module: dark_matter_profile_structure_tasks
Description: Implements structure tasks related to the dark matter halo density profile.
Code lines: 150
Contained by: file dark_matter_profiles.structure_tasks.F90
Modules used: dark_matter_profiles
Used by: function galactic_structure_density; function galactic_structure_enclosed_mass; function galactic_structure_potential; function galactic_structure_rotation_curve; function galactic_structure_rotation_curve_gradient

function: dark_matter_profile_density_task
Description: Computes the density at a given position for a dark matter profile.
Code lines: 21
Contained by: module dark_matter_profile_structure_tasks
Modules used: galactic_structure_options galacticus_nodes

function: dark_matter_profile_enclosed_mass_task
Description: Computes the mass within a given radius for a dark matter profile.
Code lines: 32
Contained by: module dark_matter_profile_structure_tasks
Modules used: galactic_structure_options galacticus_nodes

function: `dark_matter_profile_potential_task`
Description: Return the potential due to dark matter.
Code lines: 20
Contained by: module `dark_matter_profile_structure_tasks`
Modules used: `galactic_structure_options` `galacticus_error` `galacticus_nodes`

function: `dark_matter_profile_rotation_curve_gradient_task`
Description: Computes the rotation curve gradient for the dark matter.
Code lines: 24
Contained by: module `dark_matter_profile_structure_tasks`
Modules used: `galactic_structure_options` `galacticus_nodes` `numerical_constants_math` `numerical_constants_physical`

function: `dark_matter_profile_rotation_curve_task`
Description: Computes the rotation curve at a given radius for a dark matter profile.
Code lines: 20
Contained by: module `dark_matter_profile_structure_tasks`
Modules used: `galactic_structure_options` `galacticus_nodes` `numerical_constants_physical`

file: `dark_matter_profiles_DMO.Burkert.F90`
Description: An implementation of Burkert [1995] dark matter halo profiles.
Code lines: 1190
Modules used: `kind_numbers` `numerical_constants_math` `tables`

function: `burkertangularmomentumscalefree`
Description: Returns the total angular momentum (in units of the virial mass times scale radius times [assumed constant] rotation speed) in an Burkert dark matter profile with given `concentration`. This is given by:

$$J = \int_0^c 4\pi x^3 \rho(x) \mathrm{d}x \Big/ \int_0^c 4\pi x^2 \rho(x) \mathrm{d}x, \tag{18.5}$$

where x is radius in units of the scale radius and c is concentration. This can be evaluated to give

$$J = \left[2\tan^{-1} c + 2\log(1+c) + \log(1+c^2) - 4c\right] / \left[2\tan^{-1} c - 2\log(1+c) - \log(1+c^2)\right]. \tag{18.6}$$

Code lines: 17
Contained by: file `dark_matter_profiles_DMO.Burkert.F90`

subroutine: `burkertautohook`
Description: Attach to the calculation reset event.
Code lines: 8
Contained by: file `dark_matter_profiles_DMO.Burkert.F90`
Modules used: `events_hooks`

subroutine: `burkertcalculationreset`
Description: Reset the dark matter profile calculation.
Code lines: 10
Contained by: file `dark_matter_profiles_DMO.Burkert.F90`

function: `burkertcircularvelocity`
Description: Returns the circular velocity (in km/s) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).
Code lines: 15
Contained by: file `dark_matter_profiles_DMO.Burkert.F90`
Modules used: `numerical_constants_physical`

function: `burkertcircularvelocitymaximum`
Description: Returns the maximum circular velocity (in km/s) in the dark matter profile of `node`.
Code lines: 23
Contained by: file `dark_matter_profiles_DMO.Burkert.F90`
Modules used: `galacticus_nodes` `numerical_constants_physical`

function: `burkertconstructorinternal`
Description: Generic constructor for the `burkert` dark matter halo profile class.
Code lines: 33
Contained by: file `dark_matter_profiles_DMO.Burkert.F90`
Modules used: `galacticus_error` `galacticus_nodes`

function: `burkertconstructorparameters`
Description: Default constructor for the `burkert` dark matter halo profile class.
Code lines: 13
Contained by: file `dark_matter_profiles_DMO.Burkert.F90`
Modules used: `input_parameters`

function: `burkertdensity`
Description: Returns the density (in $M_\odot$ Mpc^{-3}) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).
Code lines: 19
Contained by: file `dark_matter_profiles_DMO.Burkert.F90`
Modules used: `galacticus_nodes`

function: `burkertdensitylogslope`
Description: Returns the logarithmic slope of the density profile of `node` at the given `radius` (given in units of Mpc).
Code lines: 17
Contained by: file `dark_matter_profiles_DMO.Burkert.F90`
Modules used: `galacticus_nodes`

function: `burkertdensityscalefree`
Description: Returns the density (in units such that the virial mass and scale length are unity) in an Burkert dark matter profile with given `concentration` at the given `radius` (given in units of the scale radius).
Code lines: 11
Contained by: file `dark_matter_profiles_DMO.Burkert.F90`

Modules used: numerical_constants_math

subroutine: burkertdestructor
Description: Destructor for the burkert dark matter halo profile class.
Code lines: 30
Contained by: file dark_matter_profiles_DMO.Burkert.F90
Modules used: events_hooks

function: burkertenclosedmass
Description: Returns the enclosed mass (in $M_\odot$) in the dark matter profile of node at the given radius (given in units of Mpc).
Code lines: 19
Contained by: file dark_matter_profiles_DMO.Burkert.F90
Modules used: galacticus_nodes

function: burkertenclosedmassscalefree
Description: Returns the enclosed mass (in units of the virial mass) in an Burkert dark matter profile with given concentration at the given radius (given in units of the scale radius).
Code lines: 20
Contained by: file dark_matter_profiles_DMO.Burkert.F90

function: burkertenergy
Description: Return the energy of an Burkert halo density profile.
Code lines: 23
Contained by: file dark_matter_profiles_DMO.Burkert.F90
Modules used: galacticus_nodes

function: burkertenergygrowthrate
Description: Return the rate of change of the energy of an Burkert halo density profile.
Code lines: 27
Contained by: file dark_matter_profiles_DMO.Burkert.F90
Modules used: galacticus_nodes

function: burkertfreefallradius
Description: Returns the freefall radius in the Burkert density profile at the specified time (given in Gyr).
Code lines: 37
Contained by: file dark_matter_profiles_DMO.Burkert.F90
Modules used: galacticus_nodes numerical_constants_astronomical

function: burkertfreefallradiusincreaserate
Description: Returns the rate of increase of the freefall radius in the Burkert density profile at the specified time (given in Gyr).
Code lines: 38
Contained by: file dark_matter_profiles_DMO.Burkert.F90
Modules used: galacticus_nodes numerical_constants_astronomical

subroutine: burkertfreefalltabulate
Description: Tabulates the freefall time vs. freefall radius for Burkert halos.
Code lines: 38
Contained by: file dark_matter_profiles_DMO.Burkert.F90

function: `burkertfreefalltimescalefree`
Description: Compute the freefall time in a scale-free Burkert halo.
Code lines: 46
Contained by: file `dark_matter_profiles_DMO.Burkert.F90`
Modules used: `numerical_integration`

function: `burkertfreefalltimescalefreeintegrand`
Description: Integrand function used for finding the free-fall time in Burkert halos.
Code lines: 25
Contained by: function `burkertfreefalltimescalefree`
Modules used: `numerical_constants_math`

subroutine: `burkertinverseangularmomentum`
Description: Tabulates the specific angular momentum vs. radius in an Burkert profile for rapid inversion.
Code lines: 42
Contained by: file `dark_matter_profiles_DMO.Burkert.F90`

function: `burkertkspace`
Description: Returns the Fourier transform of the Burkert density profile at the specified `waveNumber` (given in Mpc^{-1}), using the expression given in Cooray and Sheth (2002; eqn. 81).
Code lines: 28
Contained by: file `dark_matter_profiles_DMO.Burkert.F90`
Modules used: `exponential_integrals` `galacticus_nodes` `numerical_constants_math`

subroutine: `burkertmassnormalizationfactor`
Description: Compute the normalization factor for the burkert mass profile.
Code lines: 19
Contained by: file `dark_matter_profiles_DMO.Burkert.F90`

function: `burkertpotential`
Description: Returns the potential (in $(\mathrm{km/s})^2$) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).
Code lines: 25
Contained by: file `dark_matter_profiles_DMO.Burkert.F90`
Modules used: `galactic_structure_options` `galacticus_nodes` `numerical_constants_math`

function: `burkertprofileenergy`
Description: Computes the total energy of an Burkert profile halo of given `concentration` using the methods of Cole et al. (2000; their Appendix A).
Code lines: 65
Contained by: file `dark_matter_profiles_DMO.Burkert.F90`
Modules used: `numerical_constants_math` `numerical_integration`

function: `burkertjeansequationintegrand`
Description: Integrand for Burkert profile Jeans equation.
Code lines: 7

Contained by: function burkertprofileenergy

function: burkertkineticenergyintegrand
Description: Integrand for Burkert profile kinetic energy.
Code lines: 7
Contained by: function burkertprofileenergy

function: burkertpotentialenergyintegrand
Description: Integrand for Burkert profile potential energy.
Code lines: 7
Contained by: function burkertprofileenergy

function: burkertradialmoment
Description: Returns the density (in $M_\odot$ Mpc^{-3}) in the dark matter profile of node at the given radius (given in units of Mpc).
Code lines: 48
Contained by: file dark_matter_profiles_DMO.Burkert.F90
Modules used: galacticus_nodes numerical_comparison numerical_constants_math

function: radialmoment
Description: Evaluate the radial moment in the Burkert profile.
Code lines: 18
Contained by: function burkertradialmoment
Modules used: hypergeometric_functions numerical_comparison numerical_constants_math

function: burkertradialvelocitydispersion
Description: Returns the radial velocity dispersion (in km/s) in the dark matter profile of node at the given radius (given in units of Mpc).
Code lines: 26
Contained by: file dark_matter_profiles_DMO.Burkert.F90
Modules used: galacticus_nodes

function: burkertradialvelocitydispersionscalefree
Description: Compute the radial velocity dispersion in a scale-free Burkert halo.
Code lines: 33
Contained by: file dark_matter_profiles_DMO.Burkert.F90
Modules used: numerical_integration

function: burkertjeansequationintegrand
Description: Integrand for Burkert drak matter profile Jeans equation.
Code lines: 11
Contained by: function burkertradialvelocitydispersionscalefree

subroutine: burkertradialvelocitydispersiontabulate
Description: Tabulates the radial velocity dispersion vs. radius for Burkert halos.
Code lines: 33
Contained by: file dark_matter_profiles_DMO.Burkert.F90

function: `burkertradiusenclosingdensity`
Description: Null implementation of function to compute the radius enclosing a given density for Burkert dark matter halo profiles.
Code lines: 19
Contained by: file `dark_matter_profiles_DMO.Burkert.F90`
Modules used: `galacticus_nodes`

subroutine: `burkertradiusenclosingdensitytabulate`
Description: Tabulates the radius vs. enclosed density for Burkert halos.
Code lines: 41
Contained by: file `dark_matter_profiles_DMO.Burkert.F90`
Modules used: `numerical_constants_math`

function: `burkertradiusfromspecificangularmomentum`
Description: Returns the radius (in Mpc) in `node` at which a circular orbit has the given `specificAngularMomentum` (given in units of km s^{-1} Mpc)
Code lines: 46
Contained by: file `dark_matter_profiles_DMO.Burkert.F90`
Modules used: `galacticus_nodes`

function: `burkertrotationnormalization`
Description: Return the normalization of the rotation velocity vs. specific angular momentum relation.
Code lines: 21
Contained by: file `dark_matter_profiles_DMO.Burkert.F90`
Modules used: `galacticus_nodes`

function: `burkertspecificangularmomentumscalefree`
Description: Returns the specific angular momentum, normalized to unit scale length and unit velocity at the scale radius, at position `radius` (in units of the scale radius) in an Burkert profile.
Code lines: 9
Contained by: file `dark_matter_profiles_DMO.Burkert.F90`

subroutine: `burkerttabulate`
Description: Tabulate properties of the Burkert halo profile which must be computed numerically.
Code lines: 35
Contained by: file `dark_matter_profiles_DMO.Burkert.F90`

interface: `darkmatterprofiledmoburkert`
Description: Constructors for the `burkert` dark matter halo profile class.
Code lines: 4
Contained by: file `dark_matter_profiles_DMO.Burkert.F90`

file: `dark_matter_profiles_DMO.Einasto.F90`
Description: An implementation of "Einasto" dark matter halo profiles.
Code lines: 1541
Modules used: `fgsl` `kind_numbers` `tables`

interface: `darkmatterprofiledmoeinasto`
Description: Constructors for the `einasto` dark matter halo profile class.
Code lines: 4
Contained by: file dark_matter_profiles_DMO.Einasto.F90

function: `einastocircularvelocity`
Description: Returns the circular velocity (in km/s) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).
Code lines: 15
Contained by: file dark_matter_profiles_DMO.Einasto.F90
Modules used: numerical_constants_physical

function: `einastocircularvelocitymaximum`
Description: Returns the maximum circular velocity (in km/s) in the dark matter profile of `node`.
Code lines: 38
Contained by: file dark_matter_profiles_DMO.Einasto.F90
Modules used: galacticus_nodes numerical_constants_physical root_finder

function: `einastocircularvelocitypeakradius`
Description: Computes the derivative of the square of circular velocity for an Einasto density profile.
Code lines: 8
Contained by: function einastocircularvelocitymaximum
Modules used: gamma_functions

function: `einastoconstructorinternal`
Description: Internal constructor for the `einasto` dark matter halo profile class.
Code lines: 61
Contained by: file dark_matter_profiles_DMO.Einasto.F90
Modules used: array_utilities galacticus_error galacticus_nodes

function: `einastoconstructorparameters`
Description: Constructor for the `einasto` dark matter halo profile class which takes a parameter set as input.
Code lines: 13
Contained by: file dark_matter_profiles_DMO.Einasto.F90
Modules used: input_parameters

function: `einastodensity`
Description: Returns the density (in $M_\odot$ Mpc^{-3}) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).
Code lines: 21
Contained by: file dark_matter_profiles_DMO.Einasto.F90
Modules used: galacticus_nodes

function: `einastodensitylogslope`
Description: Returns the logarithmic slope of the density in the dark matter profile of `node` at the given `radius` (given in units of Mpc).

Code lines: 21
Contained by: file dark_matter_profiles_DMO.Einasto.F90
Modules used: galacticus_nodes

function: einastodensityscalefree
Description: Returns the density (in units such that the virial mass and scale length are unity) in an Einasto dark matter profile with given concentration and alpha at the given radius (given in units of the scale radius).
Code lines: 14
Contained by: file dark_matter_profiles_DMO.Einasto.F90
Modules used: gamma_functions numerical_constants_math

subroutine: einastodestructor
Description: Destructor for the einasto dark matter halo profile class.
Code lines: 19
Contained by: file dark_matter_profiles_DMO.Einasto.F90
Modules used: numerical_interpolation

function: einastoenclosedmass
Description: Returns the enclosed mass (in $M_\odot$) in the dark matter profile of node at the given radius (given in units of Mpc).
Code lines: 21
Contained by: file dark_matter_profiles_DMO.Einasto.F90
Modules used: galacticus_nodes

function: einastoenclosedmassscalefree
Description: Returns the enclosed mass (in units of the virial mass) in an Einasto dark matter profile with given concentration at the given radius (given in units of the scale radius).
Code lines: 11
Contained by: file dark_matter_profiles_DMO.Einasto.F90
Modules used: gamma_functions

function: einastoenergy
Description: Return the energy of an Einasto halo density profile.
Code lines: 41
Contained by: file dark_matter_profiles_DMO.Einasto.F90
Modules used: galacticus_nodes iso_c_binding
numerical_interpolation

function: einastoenergygrowthrate
Description: Return the energy of an Einasto halo density profile.
Code lines: 44
Contained by: file dark_matter_profiles_DMO.Einasto.F90
Modules used: galacticus_nodes iso_c_binding
numerical_interpolation

subroutine: einastoenergytablemake
Description: Create a tabulation of the energy of Einasto profiles as a function of their concentration of α parameter.
Code lines: 123

Contained by: file dark_matter_profiles_DMO.Einasto.F90
Modules used: memory_management, numerical_constants_math, numerical_integration, numerical_interpolation, numerical_ranges

function: einastojeansequationintegrand
Description: Integrand for Einasto profile Jeans equation.
Code lines: 7
Contained by: subroutine einastoenergytablemake

function: einastokineticenergyintegrand
Description: Integrand for Einasto profile kinetic energy.
Code lines: 7
Contained by: subroutine einastoenergytablemake

function: einastopotentialenergyintegrand
Description: Integrand for Einasto profile potential energy.
Code lines: 7
Contained by: subroutine einastoenergytablemake

subroutine: einastofourierprofiletablemake
Description: Create a tabulation of the Fourier transform of Einasto profiles as a function of their α parameter and dimensionless wavenumber.
Code lines: 133
Contained by: file dark_matter_profiles_DMO.Einasto.F90
Modules used: galacticus_display, galacticus_error, memory_management, numerical_integration, numerical_interpolation, numerical_ranges

function: einastofourierprofileintegrand
Description: Integrand for Einasto Fourier profile.
Code lines: 8
Contained by: subroutine einastofourierprofiletablemake
Modules used: numerical_constants_math

function: einastofreefallradius
Description: Returns the freefall radius in the Einasto density profile at the specified time (given in Gyr).
Code lines: 60
Contained by: file dark_matter_profiles_DMO.Einasto.F90
Modules used: galacticus_nodes, gamma_functions, iso_c_binding, numerical_constants_astronomical, numerical_constants_physical, numerical_interpolation

function: einastofreefallradiusincreaserate
Description: Returns the rate of increase of the freefall radius in the Einasto density profile at the specified time (given in Gyr).
Code lines: 61
Contained by: file dark_matter_profiles_DMO.Einasto.F90
Modules used: galacticus_nodes, gamma_functions, iso_c_binding, numerical_constants_astronomical

numerical_constants_physical numerical_interpolation

subroutine: `einastofreefalltabulate`
Description: Tabulates the freefall time vs. freefall radius for Einasto halos.
Code lines: 80
Contained by: file dark_matter_profiles_DMO.Einasto.F90
Modules used: galacticus_display memory_management numerical_interpolation numerical_ranges

function: `einastofreefalltimescalefree`
Description: Compute the freefall time in a scale-free Einasto halo.
Code lines: 27
Contained by: file dark_matter_profiles_DMO.Einasto.F90
Modules used: numerical_integration

function: `einastofreefalltimescalefreeintegrand`
Description: Integrand function used for finding the free-fall time in Einasto halos.
Code lines: 7
Contained by: function einastofreefalltimescalefree

function: `einastokspace`
Description: Returns the Fourier transform of the Einasto density profile at the specified `waveNumber` (given in Mpc^{-1})).
Code lines: 45
Contained by: file dark_matter_profiles_DMO.Einasto.F90
Modules used: galacticus_nodes iso_c_binding numerical_interpolation

function: `einastopotential`
Description: Returns the potential (in $(\mathrm{km/s})^2$) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).
Code lines: 26
Contained by: file dark_matter_profiles_DMO.Einasto.F90
Modules used: galactic_structure_options galacticus_nodes numerical_constants_physical

function: `einastopotentialscalefree`
Description: Returns the gravitational potential (in units where the virial mass and scale radius are unity) in an Einasto dark matter profile with given `concentration` and `alpha` at the given `radius` (given in units of the scale radius). Uses the results from Retana-Montenegro et al. [2012], their equations (19) and (20).
Code lines: 18
Contained by: file dark_matter_profiles_DMO.Einasto.F90
Modules used: gamma_functions

function: `einastoradialmoment`
Description: Returns the density (in $M_\odot$ Mpc^{-3}) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).
Code lines: 41
Contained by: file dark_matter_profiles_DMO.Einasto.F90

Modules used: galacticus_nodes gamma_functions numerical_comparison numerical_constants_math

function: einastoradialmomentscalefree
Description: Provides the scale-free part of the radial moment of the Einasto density profile.
Code lines: 7
Contained by: function einastoradialmoment

function: einastoradialvelocitydispersion
Description: Returns the radial velocity dispersion (in km/s) in the dark matter profile of node at the given radius (given in units of Mpc).
Code lines: 42
Contained by: file dark_matter_profiles_DMO.Einasto.F90
Modules used: galacticus_nodes gamma_functions iso_c_binding numerical_interpolation

function: einastoradialvelocitydispersionscalefree
Description: Compute the radial velocity dispersion in a scale-free Einasto halo.
Code lines: 34
Contained by: file dark_matter_profiles_DMO.Einasto.F90
Modules used: numerical_integration

function: einastojeansequationintegrand
Description: Integrand for Einasto drak matter profile Jeans equation.
Code lines: 12
Contained by: function einastoradialvelocitydispersionscalefree
Modules used: gamma_functions

subroutine: einastoradialvelocitydispersiontabulate
Description: Tabulates the radial velocity dispersion vs. radius for Einasto halos.
Code lines: 72
Contained by: file dark_matter_profiles_DMO.Einasto.F90
Modules used: galacticus_display memory_management numerical_interpolation numerical_ranges

function: einastoradiusenclosingdensity
Description: Implementation of function to compute the radius enclosing a given density for Einasto dark matter halo profiles. This function uses a numerical root finder to find the enclosing radius—this is likely not the most efficient solution...
Code lines: 23
Contained by: file dark_matter_profiles_DMO.Einasto.F90
Modules used: root_finder

function: einastoradiusenclosingdensityroot
Description: Root function used in finding the radius enclosing a given density in Einasto profiles.
Code lines: 8
Contained by: file dark_matter_profiles_DMO.Einasto.F90
Modules used: numerical_constants_math

function: `einastoradiusfromspecificangularmomentum`

Description: Returns the radius (in Mpc) in `node` at which a circular orbit has the given `specificAngularMomentum` (given in units of km s^{-1} Mpc).
Code lines: 23
Contained by: file dark_matter_profiles_DMO.Einasto.F90
Modules used: galacticus_nodes, numerical_constants_physical

function: `einastoradiusfromspecificangularmomentumscalefree`

Description: Comptue the radius at which a circular orbit has the given `specificAngularMomentumScaleFree` in a scale free Einasto profile.
Code lines: 32
Contained by: file dark_matter_profiles_DMO.Einasto.F90
Modules used: iso_c_binding, numerical_interpolation

subroutine: `einastoradiusfromspecificangularmomentumtablemake`

Description: Create a tabulation of the relation between specific angular momentum and radius in an Einasto profile.
Code lines: 69
Contained by: file dark_matter_profiles_DMO.Einasto.F90
Modules used: gamma_functions, memory_management, numerical_interpolation, numerical_ranges

function: `einastorotationnormalization`

Description: Return the rotation normalization of an Einasto halo density profile.
Code lines: 19
Contained by: file dark_matter_profiles_DMO.Einasto.F90
Modules used: galacticus_nodes, gamma_functions, numerical_constants_math

file: `dark_matter_profiles_DMO.F90`

Description: Contains a module which provides an object that implements dark matter halo profiles.
Code lines: 241

module: `dark_matter_profiles_dmo`

Description: Provides an object that implements dark matter halo profiles.
Code lines: 219
Contained by: file dark_matter_profiles_DMO.F90
Modules used: dark_matter_halo_scales, dark_matter_profiles_generic, galacticus_nodes
Used by: file accretion.halo.Bertschinger.F90; file cooling.cooling_rate.velocity_maximum_scaling.F90; file cooling.freefall_radii.dark_matter_halo.F90; file cooling.specific_angular_momentum.constant_rotation.F90; function dark_matter_halo_angular_momentum; function dark_matter_halo_angular_momentum_growth_rate; function dark_matter_halo_spin; file dark_matter_halos.spins.distributions.N-body_errors.F90

file dark_matter_profiles.adiabatic_-Gnedin2004.F90
file dark_matter_profiles.dark_matter_-only.F90
module dark_matter_profiles_-concentration
function dark_matter_profile_mass_-definition
file dark_matter_-profiles.structure.scale.Ludlow2016.F90
file dark_matter_-profiles.structure.scale.concentration.F90
file galactic_structure.radius_-solver.equilibrium.F90
file galactic_structure.radius_-solver.fixed.F90
file galactic_structure.radius_-solver.simple.F90
subroutine galacticus_function_-classes_destroy
file galacticus.output.analyses.correlation_-function.F90
file galacticus.output.analyses.spin_-distribution.Bett2007.F90
subroutine galacticus_extra_output_-halo_fourier_profile
subroutine galacticus_state_retrieve
subroutine galacticus_state_store
subroutine halo_model_projected_-correlation
file hot_halo.mass_-distribution.PatejLoeb2015.F90
file hot_halo.mass_-distribution.Ricotti2000.F90
file hot_halo.outflow_-reincorporation.velocity_maximum_-scaling.F90
file hot_halo.temperature_-profile.Enzo_hydrostatic.F90
file merger_trees.construct.read.F90
file merger_-trees.operators.particulate.F90
file models.likelihoods.halo_mass_-function.F90
file models.likelihoods.projected_-correlation_function.F90
file models.likelihoods.spin_-distribution.F90
file nodes.property_-extractor.spin_-Bullock.F90
file nodes.property_-extractor.velocity_maximum.F90
module node_component_disk_very_simple
module node_component_disk_very_-simple_size
module node_component_hot_halo_cold_-mode
module node_component_hot_halo_-standard
module node_component_spheroid_very_-simple
module node_component_spin_vitvitska
file satellites.dynamical_-friction.acceleration.Chandrasekhar1943.F90
file satellites.merging.timescale.Boylan-Kolchin2008.F90
file satellites.merging.timescale.Jiang2008.F90
module satellite_orbits
file star_-formation.feedback.disks.velocity_-maximum_scaling.F90
file star_-formation.feedback.spheroids.velocity_-maximum_scaling.F90
file star_-formation.timescales.disks.velocity_-maximum_scaling.F90

file star_-formation.timescales.spheroids.velocity_maximum_scaling.F90
file statistics.Nbody.halos.mass_-errors.SO_halo_finder.F90
file structure_formation.halo_mass_-function.friends_of_friends_bias.F90
module virial_density_contrast_-percolation_utilities
file tasks.halo_mass_function.F90
file tasks.halo_model.projected_-correlation_function.F90
file tasks.halo_model_generate.F90
program test_dark_matter_halo_radius_-enclosing_mass
program test_dark_matter_profiles
program test_dark_matter_profiles_-generic
program test_dark_matter_profiles_-heated

function: darkmatterprofiledmoenclosedmassroot

Description: Root function used in solving for the radius that encloses a given mass.
Code lines: 7
Contained by: module dark_matter_profiles_dmo

file: dark_matter_profiles_DMO.NFW.F90

Description: An implementation of Navarro et al. [1997] dark matter halo profiles.
Code lines: 1238
Modules used: kind_numbers tables

interface: darkmatterprofiledmonfw

Description: Constructors for the nfw dark matter halo profile class.
Code lines: 4
Contained by: file dark_matter_profiles_DMO.NFW.F90

function: nfwangularmomentumscalefree

Description: Returns the total angular momentum (in units of the virial mass times scale radius times [assumed constant] rotation speed) in an NFW dark matter profile with given concentration. This is given by:

$$J = \int_0^c 4\pi x^3 \rho(x) \mathrm{d}x \Big/ \int_0^c 4\pi x^2 \rho(x) \mathrm{d}x, \tag{18.7}$$

where x is radius in units of the scale radius and c is concentration. This can be evaluated to give

$$J = \left[1 + c - 2\ln(1+c) - \frac{1}{1+c}\right] \Big/ \left[\ln(1+c) - \frac{c}{1+c}\right]. \tag{18.8}$$

Code lines: 17
Contained by: file dark_matter_profiles_DMO.NFW.F90

subroutine: nfwautohook

Description: Attach to the calculation reset event.
Code lines: 8
Contained by: file dark_matter_profiles_DMO.NFW.F90
Modules used: events_hooks

subroutine: `nfwcalculationreset`
Description: Reset the dark matter profile calculation.
Code lines: 15
Contained by: file dark_matter_profiles_DMO.NFW.F90

function: `nfwcircularvelocity`
Description: Returns the circular velocity (in km/s) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).
Code lines: 22
Contained by: file dark_matter_profiles_DMO.NFW.F90
Modules used: numerical_constants_physical

function: `nfwcircularvelocitymaximum`
Description: Returns the maximum circular velocity (in km/s) in the dark matter profile of `node`.
Code lines: 28
Contained by: file dark_matter_profiles_DMO.NFW.F90
Modules used: galacticus_nodes numerical_constants_physical

function: `nfwconstructorinternal`
Description: Generic constructor for the `nfw` dark matter halo profile class.
Code lines: 29
Contained by: file dark_matter_profiles_DMO.NFW.F90
Modules used: galacticus_error galacticus_nodes

function: `nfwconstructorparameters`
Description: Constructor for the `nfw` dark matter halo profile class which takes a parameter set as input.
Code lines: 13
Contained by: file dark_matter_profiles_DMO.NFW.F90
Modules used: input_parameters

function: `nfwdensity`
Description: Returns the density (in $M_\odot$ Mpc^{-3}) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).
Code lines: 19
Contained by: file dark_matter_profiles_DMO.NFW.F90
Modules used: galacticus_nodes

function: `nfwdensityenclosedbyradiusscalefree`
Description: Returns the density (in units of the virial mass per cubic scale radius) in an NFW dark matter profile with given `concentration` which is enclosed a given radius (in units of the scale radius).
Code lines: 10
Contained by: file dark_matter_profiles_DMO.NFW.F90
Modules used: numerical_constants_math

function: `nfwdensitylogslope`
Description: Returns the logarithmic slope of the density in the dark matter profile of `node` at the given `radius` (given in units of Mpc).
Code lines: 19

Contained by: file dark_matter_profiles_DMO.NFW.F90
Modules used: galacticus_nodes

function: nfwdensityscalefree
Description: Returns the density (in units such that the virial mass and scale length are unity) in an NFW dark matter profile with given concentration at the given radius (given in units of the scale radius).
Code lines: 11
Contained by: file dark_matter_profiles_DMO.NFW.F90
Modules used: numerical_constants_math

subroutine: nfwdestructor
Description: Destructor for the nfw dark matter halo profile class.
Code lines: 27
Contained by: file dark_matter_profiles_DMO.NFW.F90
Modules used: events_hooks

subroutine: nfwencloseddensitytabulate
Description: Tabulates the enclosed density vs. radius for NFW halos.
Code lines: 40
Contained by: file dark_matter_profiles_DMO.NFW.F90

function: nfwenclosedmass
Description: Returns the enclosed mass (in $M_\odot$) in the dark matter profile of node at the given radius (given in units of Mpc).
Code lines: 19
Contained by: file dark_matter_profiles_DMO.NFW.F90
Modules used: galacticus_nodes

function: nfwenclosedmassscalefree
Description: Returns the enclosed mass (in units of the virial mass) in an NFW dark matter profile with given concentration at the given radius (given in units of the scale radius).
Code lines: 22
Contained by: file dark_matter_profiles_DMO.NFW.F90

function: nfwenergy
Description: Return the energy of an NFW halo density profile.
Code lines: 23
Contained by: file dark_matter_profiles_DMO.NFW.F90
Modules used: galacticus_nodes

function: nfwenergygrowthrate
Description: Return the rate of change of the energy of an NFW halo density profile.
Code lines: 26
Contained by: file dark_matter_profiles_DMO.NFW.F90
Modules used: galacticus_nodes

function: nfwfreefallradius
Description: Returns the freefall radius in the NFW density profile at the specified time (given in Gyr).
Code lines: 41

Contained by: file dark_matter_profiles_DMO.NFW.F90
Modules used: galacticus_nodes numerical_constants_astronomical

function: nfwfreefallradiusincreaserate
Description: Returns the rate of increase of the freefall radius in the NFW density profile at the specified time (given in Gyr).
Code lines: 42
Contained by: file dark_matter_profiles_DMO.NFW.F90
Modules used: galacticus_nodes numerical_constants_astronomical

subroutine: nfwfreefalltabulate
Description: Tabulates the freefall time vs. freefall radius for NFW halos.
Code lines: 40
Contained by: file dark_matter_profiles_DMO.NFW.F90

function: nfwfreefalltimescalefree
Description: Compute the freefall time in a scale-free NFW halo.
Code lines: 50
Contained by: file dark_matter_profiles_DMO.NFW.F90
Modules used: numerical_integration

function: nfwfreefalltimescalefreeintegrand
Description: Integrand function used for finding the free-fall time in NFW halos.
Code lines: 21
Contained by: function nfwfreefalltimescalefree

subroutine: nfwinverseangularmomentum
Description: Tabulates the specific angular momentum vs. radius in an NFW profile for rapid inversion.
Code lines: 42
Contained by: file dark_matter_profiles_DMO.NFW.F90

function: nfwkspace
Description: Returns the Fourier transform of the NFW density profile at the specified waveNumber (given in Mpc^{-1}), using the expression given in Cooray and Sheth (2002; eqn. 81).
Code lines: 27
Contained by: file dark_matter_profiles_DMO.NFW.F90
Modules used: exponential_integrals galacticus_nodes

subroutine: nfwmassnormalizationfactor
Description: Compute the normalization factor for the NFW mass profile.
Code lines: 19
Contained by: file dark_matter_profiles_DMO.NFW.F90

function: nfwpotential
Description: Returns the potential (in $(\mathrm{km/s})^2$) in the dark matter profile of node at the given radius (given in units of Mpc).
Code lines: 27
Contained by: file dark_matter_profiles_DMO.NFW.F90
Modules used: galactic_structure_options galacticus_nodes

function: `nfwprofileenergy`
Description: Computes the total energy of an NFW profile halo of given `concentration` using the methods of Cole et al. (2000; their Appendix A), except for potential energy which is computed using the result derived by Mo et al. (1998; eqn. 23).
Code lines: 52
Contained by: file `dark_matter_profiles_DMO.NFW.F90`
Modules used: `numerical_constants_math` `numerical_integration`

function: `nfwjeansequationintegrand`
Description: Integrand for NFW profile Jeans equation.
Code lines: 7
Contained by: function `nfwprofileenergy`

function: `nfwkineticenergyintegrand`
Description: Integrand for NFW profile kinetic energy.
Code lines: 7
Contained by: function `nfwprofileenergy`

function: `nfwradialmoment`
Description: Returns the density (in $M_\odot$ Mpc^{-3}) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).
Code lines: 48
Contained by: file `dark_matter_profiles_DMO.NFW.F90`
Modules used: `galacticus_nodes` `numerical_comparison` `numerical_constants_math`

function: `nfwradialmomentscalefree`
Description: Provides the scale-free part of the radial moment of the NFW density profile.
Code lines: 18
Contained by: function `nfwradialmoment`
Modules used: `hypergeometric_functions`

function: `nfwradialvelocitydispersion`
Description: Returns the radial velocity dispersion (in km/s) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).
Code lines: 17
Contained by: file `dark_matter_profiles_DMO.NFW.F90`
Modules used: `galacticus_nodes`

function: `nfwradialvelocitydispersionscalefree`
Description: Returns the radial velocity dispersion (in units of the virial velocity) in an NFW dark matter profile with given `concentration` at the given `radius` (given in units of the scale radius) using the result derived by Łokas and Mamon (2001; eqn. 14). Note that approximate solutions are used at small and large radii.
Code lines: 33
Contained by: file `dark_matter_profiles_DMO.NFW.F90`
Modules used: `fgsl` `numerical_constants_math`

function: `nfwradiusenclosingdensity`

Description: Returns the radius (in units of the scale radius) in an NFW dark matter profile with given `concentration` which encloses a given density (in units of the virial mass per cubic scale radius).
Code lines: 39
Contained by: file dark_matter_profiles_DMO.NFW.F90
Modules used: galacticus_nodes

function: `nfwradiusenclosingmass`
Description: Returns the radius (in Mpc) in an NFW dark matter profile with given `concentration` which encloses a given mass (in $M_\odot$).
Code lines: 40
Contained by: file dark_matter_profiles_DMO.NFW.F90
Modules used: fgsl galacticus_nodes

function: `nfwradiusfromspecificangularmomentum`
Description: Returns the radius (in Mpc) in `node` at which a circular orbit has the given `specificAngularMomentum` (given in units of km s^{-1} Mpc).
Code lines: 45
Contained by: file dark_matter_profiles_DMO.NFW.F90
Modules used: galacticus_nodes

function: `nfwrotationnormalization`
Description: Return the normalization of the rotation velocity vs. specific angular momentum relation.
Code lines: 21
Contained by: file dark_matter_profiles_DMO.NFW.F90
Modules used: galacticus_nodes

function: `nfwspecificangularmomentumscalefree`
Description: Returns the specific angular momentum, normalized to unit scale length and unit velocity at the scale radius, at position `radius` (in units of the scale radius) in an NFW profile.
Code lines: 17
Contained by: file dark_matter_profiles_DMO.NFW.F90

subroutine: `nfwtabulate`
Description: Tabulate properties of the NFW halo profile which must be computed numerically.
Code lines: 35
Contained by: file dark_matter_profiles_DMO.NFW.F90

file: `dark_matter_profiles_DMO.heated.F90`
Description: An implementation of heated dark matter halo profiles.
Code lines: 512
Modules used: dark_matter_profiles_generic kind_numbers

interface: `darkmatterprofiledmoheated`
Description: Constructors for the `heated` dark matter halo profile class.
Code lines: 4
Contained by: file dark_matter_profiles_DMO.heated.F90

subroutine: `heatedautohook`

Description: Attach to the calculation reset event.
Code lines: 8
Contained by: file dark_matter_profiles_DMO.heated.F90
Modules used: events_hooks

subroutine: heatedcalculationreset
Description: Reset the dark matter profile calculation.
Code lines: 10
Contained by: file dark_matter_profiles_DMO.heated.F90

function: heatedcircularvelocity
Description: Returns the circular velocity (in km/s) in the dark matter profile of node at the given radius (given in units of Mpc).
Code lines: 15
Contained by: file dark_matter_profiles_DMO.heated.F90
Modules used: numerical_constants_physical

function: heatedcircularvelocitymaximum
Description: Returns the maximum circular velocity (in km/s) in the dark matter profile of node.
Code lines: 12
Contained by: file dark_matter_profiles_DMO.heated.F90

function: heatedconstructorinternal
Description: Generic constructor for the heated dark matter profile class.
Code lines: 17
Contained by: file dark_matter_profiles_DMO.heated.F90
Modules used: galacticus_error

function: heatedconstructorparameters
Description: Default constructor for the heated dark matter halo profile class.
Code lines: 28
Contained by: file dark_matter_profiles_DMO.heated.F90
Modules used: input_parameters

function: heateddensity
Description: Returns the density (in $M_\odot$ Mpc^{-3}) in the dark matter profile of node at the given radius (given in units of Mpc).
Code lines: 18
Contained by: file dark_matter_profiles_DMO.heated.F90
Modules used: galacticus_display numerical_constants_math numerical_constants_physical

function: heateddensitylogslope
Description: Returns the logarithmic slope of the density in the dark matter profile of node at the given radius (given in units of Mpc).
Code lines: 14
Contained by: file dark_matter_profiles_DMO.heated.F90

subroutine: heateddestructor

Description: Destructor for the `heated` dark matter halo profile class.
Code lines: 11
Contained by: file `dark_matter_profiles_DMO.heated.F90`
Modules used: `events_hooks`

function: `heatedenclosedmass`
Description: Returns the enclosed mass (in $M_\odot$) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).
Code lines: 10
Contained by: file `dark_matter_profiles_DMO.heated.F90`

function: `heatedenergy`
Description: Return the energy of a heated halo density profile.
Code lines: 12
Contained by: file `dark_matter_profiles_DMO.heated.F90`

function: `heatedenergygrowthrate`
Description: Return the rate of change of the energy of a heated halo density profile.
Code lines: 12
Contained by: file `dark_matter_profiles_DMO.heated.F90`

function: `heatedfreefallradius`
Description: Returns the freefall radius in the heated density profile at the specified `time` (given in Gyr).
Code lines: 14
Contained by: file `dark_matter_profiles_DMO.heated.F90`

function: `heatedfreefallradiusincreaserate`
Description: Returns the rate of increase of the freefall radius in the heated density profile at the specified `time` (given in Gyr).
Code lines: 14
Contained by: file `dark_matter_profiles_DMO.heated.F90`

function: `heatedkspace`
Description: Returns the Fourier transform of the heated density profile at the specified `waveNumber` (given in Mpc^{-1}), using the expression given in Cooray and Sheth (2002; eqn. 81).
Code lines: 14
Contained by: file `dark_matter_profiles_DMO.heated.F90`

function: `heatedpotential`
Description: Returns the potential (in $(\mathrm{km/s})^2$) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).
Code lines: 15
Contained by: file `dark_matter_profiles_DMO.heated.F90`

function: `heatedradialmoment`
Description: Returns the density (in $M_\odot$ Mpc^{-3}) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).
Code lines: 15
Contained by: file `dark_matter_profiles_DMO.heated.F90`

function: `heatedradialvelocitydispersion`
Description: Returns the radial velocity dispersion (in km/s) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).
Code lines: 18
Contained by: file dark_matter_profiles_DMO.heated.F90

function: `heatedradiusenclosingdensity`
Description: Returns the radius (in Mpc) in the dark matter profile of `node` which encloses the given `density` (given in units of $M_\odot/\mathrm{Mpc}^{-3}$).
Code lines: 14
Contained by: file dark_matter_profiles_DMO.heated.F90

function: `heatedradiusenclosingmass`
Description: Returns the radius (in Mpc) in the dark matter profile of `node` which encloses the given `mass` (given in units of $M_\odot$).
Code lines: 18
Contained by: file dark_matter_profiles_DMO.heated.F90
Modules used: galactic_structure_options numerical_constants_physical

function: `heatedradiusfromspecificangularmomentum`
Description: Returns the radius (in Mpc) in `node` at which a circular orbit has the given `specificAngularMomentum` (given in units of km s^{-1} Mpc).
Code lines: 14
Contained by: file dark_matter_profiles_DMO.heated.F90

function: `heatedradiusinitial`
Description: Find the initial radius corresponding to the given `radiusFinal` in the heated dark matter profile.
Code lines: 51
Contained by: file dark_matter_profiles_DMO.heated.F90
Modules used: root_finder

function: `heatedradiusinitialroot`
Description: Root function used in finding initial radii in heated dark matter halo profiles.
Code lines: 10
Contained by: file dark_matter_profiles_DMO.heated.F90
Modules used: numerical_constants_physical

function: `heatedrotationnormalization`
Description: Return the normalization of the rotation velocity vs. specific angular momentum relation.
Code lines: 12
Contained by: file dark_matter_profiles_DMO.heated.F90

file: `dark_matter_profiles_DMO.isothermal.F90`
Description: An implementation of isothermal dark matter halo profiles.
Code lines: 342

interface: `darkmatterprofiledmoisothermal`
Description: Constructors for the `isothermal` dark matter halo profile class.

Code lines: 4
Contained by: file dark_matter_profiles_DMO.isothermal.F90

function: isothermalcircularvelocity
Description: Returns the circular velocity (in km/s) in the dark matter profile of node at the given radius (given in units of Mpc). For an isothermal halo this is independent of radius and therefore equal to the virial velocity.
Code lines: 11
Contained by: file dark_matter_profiles_DMO.isothermal.F90

function: isothermalcircularvelocitymaximum
Description: Returns the maximum circular velocity (in km/s) in the dark matter profile of node. For an isothermal halo circular velocity is independent of radius.
Code lines: 9
Contained by: file dark_matter_profiles_DMO.isothermal.F90

function: isothermalconstructorinternal
Description: Generic constructor for the isothermal dark matter halo profile class.
Code lines: 9
Contained by: file dark_matter_profiles_DMO.isothermal.F90
Modules used: input_parameters

function: isothermalconstructorparameters
Description: Default constructor for the isothermal dark matter halo profile class.
Code lines: 13
Contained by: file dark_matter_profiles_DMO.isothermal.F90
Modules used: input_parameters

function: isothermaldensity
Description: Returns the density (in $M_\odot$ Mpc^{-3}) in the dark matter profile of node at the given radius (given in units of Mpc).
Code lines: 14
Contained by: file dark_matter_profiles_DMO.isothermal.F90
Modules used: galacticus_nodes numerical_constants_math

function: isothermaldensitylogslope
Description: Returns the logarithmic slope of the density in the dark matter profile of node at the given radius (given in units of Mpc).
Code lines: 11
Contained by: file dark_matter_profiles_DMO.isothermal.F90

subroutine: isothermaldestructor
Description: Destructor for the isothermal dark matter halo profile class.
Code lines: 7
Contained by: file dark_matter_profiles_DMO.isothermal.F90

function: isothermalenclosedmass
Description: Returns the enclosed mass (in $M_\odot$) in the dark matter profile of node at the given radius (given in units of Mpc).
Code lines: 13

Contained by: file dark_matter_profiles_DMO.isothermal.F90
Modules used: galacticus_nodes

function: isothermalenergy
Description: Return the energy of an isothermal halo density profile.
Code lines: 11
Contained by: file dark_matter_profiles_DMO.isothermal.F90
Modules used: galacticus_nodes

function: isothermalenergygrowthrate
Description: Return the rate of change of the energy of an isothermal halo density profile.
Code lines: 11
Contained by: file dark_matter_profiles_DMO.isothermal.F90
Modules used: galacticus_nodes

function: isothermalfreefallradius
Description: Returns the freefall radius in the isothermal density profile at the specified `time` (given in Gyr). For an isothermal potential, the freefall radius, $r_{\rm ff}(t)$, is:

$$r_{\rm ff}(t) = \sqrt{\frac{2}{\pi}} V_{\rm virial} t. \tag{18.9}$$

Code lines: 14
Contained by: file dark_matter_profiles_DMO.isothermal.F90
Modules used: numerical_constants_astronomical

function: isothermalfreefallradiusincreaserate
Description: Returns the rate of increase of the freefall radius in the isothermal density profile at the specified `time` (given in Gyr). For an isothermal potential, the rate of increase of the freefall radius, $\dot{r}_{\rm ff}(t)$, is:

$$\dot{r}_{\rm ff}(t) = \sqrt{\frac{2}{\pi}} V_{\rm virial}. \tag{18.10}$$

Code lines: 15
Contained by: file dark_matter_profiles_DMO.isothermal.F90
Modules used: numerical_constants_astronomical

function: isothermalkspace
Description: Returns the Fourier transform of the isothermal density profile at the specified `waveNumber` (given in Mpc^{-1}), using the expression given in Cooray and Sheth (2002; table 1).
Code lines: 20
Contained by: file dark_matter_profiles_DMO.isothermal.F90
Modules used: exponential_integrals

function: isothermalpotential
Description: Returns the potential (in (km/s)^2) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).
Code lines: 22
Contained by: file dark_matter_profiles_DMO.isothermal.F90
Modules used: galactic_structure_options galacticus_error

function: isothermalradialmoment

Description: Returns the density (in $M_\odot$ Mpc^{-3}) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).
Code lines: 25
Contained by: file dark_matter_profiles_DMO.isothermal.F90
Modules used: galacticus_nodes numerical_comparison numerical_constants_math

function: `isothermalradialvelocitydispersion`
Description: Returns the radial velocity dispersion (in km/s) in the dark matter profile of `node` at the given `radius` (given in units of Mpc). For an isothermal halo this is independent of radius and equal to the virial velocity divided by $\sqrt{(2)}$.
Code lines: 11
Contained by: file dark_matter_profiles_DMO.isothermal.F90

function: `isothermalradiusenclosingdensity`
Description: Null implementation of function to compute the radius enclosing a given density for isothermal dark matter halo profiles.
Code lines: 13
Contained by: file dark_matter_profiles_DMO.isothermal.F90
Modules used: galacticus_nodes numerical_constants_math

function: `isothermalradiusfromspecificangularmomentum`
Description: Returns the radius (in Mpc) in `node` at which a circular orbit has the given `specificAngularMomentum` (given in units of km s^{-1} Mpc). For an isothermal halo, the circular velocity is constant (and therefore equal to the virial velocity). Therefore, $r = j/V_{\rm virial}$ where j(=`specificAngularMomentum`) is the specific angular momentum and r the required radius.
Code lines: 12
Contained by: file dark_matter_profiles_DMO.isothermal.F90

function: `isothermalrotationnormalization`
Description: Return the normalization of the rotation velocity vs. specific angular momentum relation.
Code lines: 8
Contained by: file dark_matter_profiles_DMO.isothermal.F90

file: `dark_matter_profiles_DMO.truncated.F90`
Description: An implementation of truncated dark matter halo profiles.
Code lines: 497
Modules used: dark_matter_profiles_generic

interface: `darkmatterprofiledmotruncated`
Description: Constructors for the `truncated` dark matter halo profile class.
Code lines: 4
Contained by: file dark_matter_profiles_DMO.truncated.F90

subroutine: `truncatedautohook`
Description: Attach to the calculation reset event.
Code lines: 8
Contained by: file dark_matter_profiles_DMO.truncated.F90

Modules used: events_hooks

subroutine: `truncatedcalculationreset`
Description: Reset the dark matter profile calculation.
Code lines: 13
Contained by: file dark_matter_profiles_DMO.truncated.F90

function: `truncatedcircularvelocity`
Description: Returns the circular velocity (in km/s) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).
Code lines: 14
Contained by: file dark_matter_profiles_DMO.truncated.F90

function: `truncatedcircularvelocitymaximum`
Description: Returns the maximum circular velocity (in km/s) in the dark matter profile of `node`.
Code lines: 12
Contained by: file dark_matter_profiles_DMO.truncated.F90

function: `truncatedconstructorinternal`
Description: Internal constructor for the `truncated` dark matter profile class.
Code lines: 15
Contained by: file dark_matter_profiles_DMO.truncated.F90
Modules used: galacticus_error

function: `truncatedconstructorparameters`
Description: Constructor for the `truncated` dark matter halo profile class which takes a parameter set as input.
Code lines: 42
Contained by: file dark_matter_profiles_DMO.truncated.F90
Modules used: input_parameters

function: `truncateddensity`
Description: Returns the density (in $M_\odot$ Mpc^{-3}) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).
Code lines: 12
Contained by: file dark_matter_profiles_DMO.truncated.F90

function: `truncateddensitylogslope`
Description: Returns the logarithmic slope of the density in the dark matter profile of `node` at the given `radius` (given in units of Mpc).
Code lines: 16
Contained by: file dark_matter_profiles_DMO.truncated.F90

subroutine: `truncateddestructor`
Description: Destructor for the `truncated` dark matter halo profile class.
Code lines: 10
Contained by: file dark_matter_profiles_DMO.truncated.F90
Modules used: events_hooks

function: `truncatedenclosedmass`
Description: Returns the enclosed mass (in $M_\odot$) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).
Code lines: 18
Contained by: file `dark_matter_profiles_DMO.truncated.F90`

function: `truncatedenergy`
Description: Return the energy of a truncated halo density profile.
Code lines: 12
Contained by: file `dark_matter_profiles_DMO.truncated.F90`

function: `truncatedenergygrowthrate`
Description: Return the rate of change of the energy of a truncated halo density profile.
Code lines: 12
Contained by: file `dark_matter_profiles_DMO.truncated.F90`

function: `truncatedfreefallradius`
Description: Returns the freefall radius in the truncated density profile at the specified `time` (given in Gyr).
Code lines: 14
Contained by: file `dark_matter_profiles_DMO.truncated.F90`

function: `truncatedfreefallradiusincreaserate`
Description: Returns the rate of increase of the freefall radius in the truncated density profile at the specified `time` (given in Gyr).
Code lines: 14
Contained by: file `dark_matter_profiles_DMO.truncated.F90`

function: `truncatedkspace`
Description: Returns the Fourier transform of the truncated density profile at the specified `waveNumber` (given in Mpc^{-1}).
Code lines: 14
Contained by: file `dark_matter_profiles_DMO.truncated.F90`

function: `truncatedpotential`
Description: Returns the potential (in $(\mathrm{km/s})^2$) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).
Code lines: 15
Contained by: file `dark_matter_profiles_DMO.truncated.F90`

function: `truncatedradialmoment`
Description: Returns the density (in $M_\odot$ Mpc^{-3}) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).
Code lines: 15
Contained by: file `dark_matter_profiles_DMO.truncated.F90`

function: `truncatedradialvelocitydispersion`
Description: Returns the radial velocity dispersion (in km/s) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).
Code lines: 26

Contained by: file dark_matter_profiles_DMO.truncated.F90

function: truncatedradiusenclosingdensity
Description: Returns the radius (in Mpc) in the dark matter profile of node which encloses the given density (given in units of $M_\odot/\mathrm{Mpc}^{-3}$).
Code lines: 14
Contained by: file dark_matter_profiles_DMO.truncated.F90

function: truncatedradiusenclosingmass
Description: Returns the radius (in Mpc) in the dark matter profile of node which encloses the given mass (given in units of $M_\odot$).
Code lines: 21
Contained by: file dark_matter_profiles_DMO.truncated.F90

function: truncatedradiusfromspecificangularmomentum
Description: Returns the radius (in Mpc) in node at which a circular orbit has the given specificAngularMomentum (given in units of km s^{-1} Mpc).
Code lines: 14
Contained by: file dark_matter_profiles_DMO.truncated.F90

function: truncatedrotationnormalization
Description: Return the normalization of the rotation velocity vs. specific angular momentum relation.
Code lines: 12
Contained by: file dark_matter_profiles_DMO.truncated.F90

subroutine: truncatedtruncationfunction
Description: Return the scaled truncation radial coordinate, and the truncation multiplier.
Code lines: 26
Contained by: file dark_matter_profiles_DMO.truncated.F90

file: dark_matter_profiles_DMO.truncated.exponential.F90
Description: An implementation of exponentially truncated dark matter halo profiles Kazantzidis et al. [2006].
Code lines: 529
Modules used: dark_matter_profiles_generic

interface: darkmatterprofiledmotruncatedexponential
Description: Constructors for the exponentially truncated dark matter halo profile class.
Code lines: 4
Contained by: file dark_matter_profiles_DMO.truncated.exponential.F90

subroutine: recomputekappa
Description: Recompute parameter kappa in the truncation funciton.
Code lines: 16
Contained by: file dark_matter_profiles_DMO.truncated.exponential.F90
Modules used: galacticus_nodes

subroutine: truncatedexponentialautohook
Description: Attach to the calculation reset event.

Code lines: 8
Contained by: file dark_matter_profiles_DMO.truncated.exponential.F90
Modules used: events_hooks

subroutine: truncatedexponentialcalculationreset
Description: Reset the dark matter profile calculation.
Code lines: 10
Contained by: file dark_matter_profiles_DMO.truncated.exponential.F90

function: truncatedexponentialcircularvelocity
Description: Returns the circular velocity (in km/s) in the dark matter profile of node at the given radius (given in units of Mpc).
Code lines: 14
Contained by: file dark_matter_profiles_DMO.truncated.exponential.F90

function: truncatedexponentialcircularvelocitymaximum
Description: Returns the maximum circular velocity (in km/s) in the dark matter profile of node.
Code lines: 12
Contained by: file dark_matter_profiles_DMO.truncated.exponential.F90

function: truncatedexponentialconstructorinternal
Description: Internal constructor for the exponentially truncated dark matter profile class.
Code lines: 15
Contained by: file dark_matter_profiles_DMO.truncated.exponential.F90
Modules used: galacticus_error

function: truncatedexponentialconstructorparameters
Description: Constructor for the exponentially truncated dark matter halo profile class which takes a parameter set as input.
Code lines: 59
Contained by: file dark_matter_profiles_DMO.truncated.exponential.F90
Modules used: galacticus_error input_parameters

function: truncatedexponentialdensity
Description: Returns the density (in $M_\odot$ Mpc^{-3}) in the dark matter profile of node at the given radius (given in units of Mpc).
Code lines: 12
Contained by: file dark_matter_profiles_DMO.truncated.exponential.F90

function: truncatedexponentialdensitylogslope
Description: Returns the logarithmic slope of the density in the dark matter profile of node at the given radius (given in units of Mpc).
Code lines: 17
Contained by: file dark_matter_profiles_DMO.truncated.exponential.F90
Modules used: galacticus_error

subroutine: truncatedexponentialdestructor
Description: Destructor for the exponentially truncated dark matter halo profile class.
Code lines: 10

Contained by: file dark_matter_profiles_DMO.truncated.exponential.F90
Modules used: events_hooks

function: truncatedexponentialenclosedmass
Description: Returns the enclosed mass (in $M_\odot$) in the dark matter profile of node at the given radius (given in units of Mpc).
Code lines: 21
Contained by: file dark_matter_profiles_DMO.truncated.exponential.F90
Modules used: fgsl numerical_constants_math

function: truncatedexponentialenergy
Description: Return the energy of a truncatedExponential halo density profile.
Code lines: 12
Contained by: file dark_matter_profiles_DMO.truncated.exponential.F90

function: truncatedexponentialenergygrowthrate
Description: Return the rate of change of the energy of a truncatedExponential halo density profile.
Code lines: 12
Contained by: file dark_matter_profiles_DMO.truncated.exponential.F90

function: truncatedexponentialfreefallradius
Description: Returns the freefall radius in the truncatedExponential density profile at the specified time (given in Gyr).
Code lines: 14
Contained by: file dark_matter_profiles_DMO.truncated.exponential.F90

function: truncatedexponentialfreefallradiusincreaserate
Description: Returns the rate of increase of the freefall radius in the truncatedExponential density profile at the specified time (given in Gyr).
Code lines: 14
Contained by: file dark_matter_profiles_DMO.truncated.exponential.F90

function: truncatedexponentialkspace
Description: Returns the Fourier transform of the truncatedExponential density profile at the specified waveNumber (given in Mpc^{-1}).
Code lines: 14
Contained by: file dark_matter_profiles_DMO.truncated.exponential.F90

function: truncatedexponentialpotential
Description: Returns the potential (in $(\mathrm{km/s})^2$) in the dark matter profile of node at the given radius (given in units of Mpc).
Code lines: 15
Contained by: file dark_matter_profiles_DMO.truncated.exponential.F90

function: truncatedexponentialradialmoment
Description: Returns the density (in $M_\odot$ Mpc^{-3}) in the dark matter profile of node at the given radius (given in units of Mpc).
Code lines: 15
Contained by: file dark_matter_profiles_DMO.truncated.exponential.F90

function: `truncatedexponentialradialvelocitydispersion`
Description: Returns the radial velocity dispersion (in km/s) in the dark matter profile of `node` at the given `radius` (given in units of Mpc).
Code lines: 25
Contained by: file dark_matter_profiles_DMO.truncated.exponential.F90

function: `truncatedexponentialradiusenclosingdensity`
Description: Returns the radius (in Mpc) in the dark matter profile of `node` which encloses the given `density` (given in units of $M_\odot/\mathrm{Mpc}^{-3}$).
Code lines: 14
Contained by: file dark_matter_profiles_DMO.truncated.exponential.F90

function: `truncatedexponentialradiusenclosingmass`
Description: Returns the radius (in Mpc) in the dark matter profile of `node` which encloses the given `mass` (given in units of $M_\odot$).
Code lines: 19
Contained by: file dark_matter_profiles_DMO.truncated.exponential.F90
Modules used: galacticus_nodes

function: `truncatedexponentialradiusfromspecificangularmomentum`
Description: Returns the radius (in Mpc) in `node` at which a circular orbit has the given `specificAngularMomentum` (given in units of km s^{-1} Mpc).
Code lines: 14
Contained by: file dark_matter_profiles_DMO.truncated.exponential.F90

function: `truncatedexponentialrotationnormalization`
Description: Return the normalization of the rotation velocity vs. specific angular momentum relation.
Code lines: 12
Contained by: file dark_matter_profiles_DMO.truncated.exponential.F90

subroutine: `truncatedexponentialtruncationfunction`
Description: Return the scaled truncation radial coordinate, and the truncation multiplier.
Code lines: 22
Contained by: file dark_matter_profiles_DMO.truncated.exponential.F90

file: `events.branch_jump.F90`
Description: Contains a module which handles node branch jump events.
Code lines: 105

module: `node_branch_jumps`
Description: Handles satellite node branch jump events.
Code lines: 83
Contained by: file events.branch_jump.F90
Used by: subroutine readcreatebranchjumpevent

function: `node_branch_jump`
Description: Moves a satellite node to a different branch of the merger tree.
Code lines: 73

Contained by: module node_branch_jumps
Modules used: galacticus_display, galacticus_nodes, iso_varying_string, merger_trees_evolve_deadlock_status, node_component_satellite_preset, string_handling

file: events.halo_formation.F90
Description: Contains a module which performs tasks associated with "halo formation" events.
Code lines: 73

module: events_halo_formation
Description: Performs tasks associated with "halo formation" events.
Code lines: 51
Contained by: file events.halo_formation.F90
Used by: subroutine node_component_formation_-time_cole2000_create

subroutine: event_halo_formation
Description: Perform tasks associated with a "halo formation" event in thisNode.
Code lines: 41
Contained by: module events_halo_formation
Modules used: galacticus_nodes, node_component_hot_halo_cold_mode, node_component_hot_halo_standard, node_component_satellite_standard, node_component_satellite_very_simple

file: events.hooks.F90
Description: Contains module which handles hooking of objects into events.
Code lines: 294

module: events_hooks
Description: Handles hooking of object function class into events.
Code lines: 272
Contained by: file events.hooks.F90
Used by: program galacticus, subroutine naozbarkana2007autohook, subroutine naozbarkana2007destructor, subroutine coldmodeautohook, subroutine coldmodedestructor, subroutine betaprofileautohook, subroutine betaprofiledestructor, subroutine isothermalautohook, subroutine isothermaldestructor, subroutine simpleautohook, subroutine simpledestructor, subroutine constantrotationautohook, subroutine constantrotationdestructor, subroutine virialdensitycontrastdefinitionautohook, subroutine virialdensitycontrastdefinitiondestructor, subroutine adiabaticgnedin2004autohook, subroutine adiabaticgnedin2004destructor, subroutine burkertautohook, subroutine burkertdestructor, subroutine nfwautohook, subroutine nfwdestructor, subroutine heatedautohook, subroutine heateddestructor, subroutine truncatedautohook, subroutine truncateddestructor, subroutine truncatedexponentialautohook

subroutine truncatedexponentialdestructor
subroutine equilibriumautohook
subroutine equilibriumdestructor
subroutine fixedautohook
subroutine fixeddestructor
subroutine linearautohook
subroutine lineardestructor
subroutine simpleautohook
subroutine simpledestructor
subroutine galacticus_output_close_file
subroutine velocitymaximumscalingautohook
subroutine velocitymaximumscalingdestructor
subroutine internalautohook
subroutine historyautohook
subroutine recordevolutionautohook
subroutine standardoutput
subroutine intergalacticbackgroundinternalautohook
subroutine zentner2005autohook
subroutine zentner2005destructor
subroutine insituautohook
subroutine blitz2006autohook
subroutine blitz2006destructor
subroutine kennicuttschmidtautohook
subroutine kennicuttschmidtdestructor
subroutine krumholz2009autohook
subroutine krumholz2009destructor
subroutine extendedschmidtautohook
subroutine extendedschmidtdestructor
subroutine haloscalingautohook
subroutine haloscalingdestructor
subroutine velocitymaxscalingautohook
subroutine velocitymaxscalingdestructor
subroutine velocitymaxscalingautohook
subroutine velocitymaxscalingdestructor
subroutine evolveforestsautohook
subroutine evolveforestsdestructor
subroutine evolveforestsperform
program test_diemerkravtsov2014_concentration
program test_nfw96_concentration_dark_energy
program test_prada2011_concentration
program test_zhao2009_flat
program test_zhao2009_dark_energy
program test_zhao2009_open
program test_correa2015_concentration
program test_dark_matter_halo_radius_enclosing_mass
program test_dark_matter_profiles
program test_dark_matter_profiles_generic
program test_dark_matter_profiles_heated
program test_correa2015_mah
program test_stellar_populations_luminosities

type: `eventhook`
Description: Class used to define a set of hooked function calls for a given event.
Code lines: 47
Contained by: module events_hooks

function: `eventhookcount`
Description: Return a count of the number of hooks into this event.
Code lines: 11
Contained by: module events_hooks
Modules used: galacticus_error

subroutine: `eventhookdestructor`
Description: Destructor for event hook class.
Code lines: 6

Contained by: module events_hooks

function: `eventhookfirst`
Description: Return a pointer to the first hook into this event.
Code lines: 10
Contained by: module events_hooks
Modules used: galacticus_error

subroutine: `eventhookinitialize`
Description: Initialize the OpenMP lock in an event object.
Code lines: 9
Contained by: module events_hooks

subroutine: `eventhooklock`
Description: Lock the event to avoid race conditions between OpenMP threads.
Code lines: 8
Contained by: module events_hooks

subroutine: `eventhookunlock`
Description: Unlock the event to avoid race conditions between OpenMP threads.
Code lines: 8
Contained by: module events_hooks

type: `eventhookunspecified`
Description: Class used to define a set of hooked function calls for a given event.
Code lines: 21
Contained by: module events_hooks

subroutine: `eventhookunspecifiedattach`
Description: Attach an object to an event hook.
Code lines: 46
Contained by: module events_hooks
Modules used: galacticus_error

subroutine: `eventhookunspecifieddetach`
Description: Attach an object to an event hook.
Code lines: 37
Contained by: module events_hooks
Modules used: galacticus_error

type: `hook`
Description: Base class for individual hooked function calls. Stores the object to be passed as the first argument to the function.
Code lines: 6
Contained by: module events_hooks

type: `hookunspecified`
Description: Class for hooked function calls with unspecified interfaces.
Code lines: 3
Contained by: module events_hooks

file: `events.inter_tree.F90`
Description: Contains a module which handles inter-tree nodes events.
Code lines: 571

module: `node_events_inter_tree`
Description: Handles inter-tree node events.
Code lines: 549
Contained by: file `events.inter_tree.F90`
Modules used: `galacticus_nodes` `iso_c_binding` `kind_numbers`
Used by: subroutine `readassignsplitforestevents` subroutine `evolveforestsperform`

subroutine: `inter_tree_event_post_evolve`
Description: Check that the inter-tree transfer list is empty after universe evolution.
Code lines: 23
Contained by: module `node_events_inter_tree`
Modules used: `galacticus_display` `galacticus_error` `iso_varying_string` `string_handling`

type: `intertreetransfer`
Description: Type used for transfering nodes between trees.
Code lines: 6
Contained by: module `node_events_inter_tree`

function: `node_pull_from_tree`
Description: Pull a node from the tree.
Code lines: 372
Contained by: module `node_events_inter_tree`
Modules used: `galacticus_display` `galacticus_error` `galacticus_nodes` `iso_varying_string` `merger_trees_evolve_deadlock_status` `node_component_satellite_preset` `string_handling`

function: `node_push_from_tree`
Description: Push a node from the tree.
Code lines: 119
Contained by: module `node_events_inter_tree`
Modules used: `galacticus_display` `galacticus_error` `galacticus_nodes` `iso_varying_string` `merger_trees_evolve_deadlock_status` `string_handling`

file: `events.node_promotion.index_shift.F90`
Description: Contains a module which optionally shifts the index of a node about to be promoted to its parent node, allowing indices to be tracked along merger trees.
Code lines: 79

module: `node_promotion_index_shifts`
Description: Implements optional shifting of the index of a node about to be promoted to its parent node, allowing indices to be tracked along merger trees.

Code lines: 56
Contained by: file events.node_promotion.index_shift.F90
Used by: subroutine standardpromote

subroutine: node_promotion_index_shift
Description: Shifts the index of thisNode to its parent node just prior to promotion, thereby allowing indices to track galaxies through the tree.
Code lines: 36
Contained by: module node_promotion_index_shifts
Modules used: galacticus_nodes input_parameters

file: events.satellite_merger.remnant_properties.F90
Description: Contains a module which computes remnant properties for satellite mergers.
Code lines: 59

module: satellite_merging_remnant_properties
Description: Computes remnant properties for satellite mergers.
Code lines: 37
Contained by: file events.satellite_merger.remnant_properties.F90
Used by: subroutine satellitemergerprocess
subroutine node_component_age_statistics_standard_satellite_merging
subroutine node_component_disk_standard_satellite_merging
subroutine node_component_disk_very_simple_satellite_merging
subroutine node_component_inter_output_standard_satellite_merging
subroutine node_component_merging_statistics_major_satellite_merging
subroutine node_component_merging_statistics_standard_satellite_merging
subroutine node_component_spheroid_standard_satellite_merging
subroutine node_component_spheroid_very_simple_satellite_merging

subroutine: satellite_merging_remnant_compute
Description: Compute properties for satellite merger remnants.
Code lines: 18
Contained by: module satellite_merging_remnant_properties
Modules used: galacticus_nodes satellite_merging_mass_movements satellite_merging_remnant_sizes

file: events.subhalo_promotion.F90
Description: Contains a module which handles node subhalo promotion events.
Code lines: 114

module: node_subhalo_promotions
Description: Handles subhalo promotion events.
Code lines: 92
Contained by: file events.subhalo_promotion.F90
Used by: subroutine readscanforsubhalopromotions

function: node_subhalo_promotion
Description: Promotes a subhalo to be an isolated node.

Code lines: 82
Contained by: module node_subhalo_promotions
Modules used: galacticus_display, galacticus_nodes, iso_varying_string, merger_trees_evolve_deadlock_status, merger_trees_evolve_node, node_component_satellite_preset, string_handling

file: `fftw3_config.F90`
Description: Contains a test code used to see if FFTW3 is available.
Code lines: 29

program: `fftw3_config`
Description: Test code used to see if FFTW3 is available.
Code lines: 4
Contained by: file fftw3_config.F90
Modules used: iso_c_binding

file: `functions.global.pointers.F90`
Description: Contains a module providing pointers to global functions.
Code lines: 184

module: `functions_global`
Description: Provides pointers to global functions.
Code lines: 162
Contained by: file functions.global.pointers.F90
Used by: function genericenergyevaluate, subroutine functions_global_set, function hivshalomassrelationpadmanabhan2017constructorparameters, function spindistributionbett2007constructorparameters, function buildconstruct, function readconstruct, subroutine mergertreestatestore, function staterestoredconstruct, function galaxypopulationevaluate, function percolationconstructorparameters, subroutine percolationdeepcopy, subroutine percolationtabulate

subroutine: `galacticus_calculations_reset_null`
Code lines: 11
Contained by: module functions_global
Modules used: galacticus_error, galacticus_nodes

subroutine: `galacticus_state_retrieve_null`
Code lines: 5
Contained by: module functions_global
Modules used: galacticus_error

subroutine: `galacticus_state_store_null`
Code lines: 11
Contained by: module functions_global
Modules used: galacticus_error, iso_varying_string

subroutine: `percolationobjectsdeepcopy_null`
Code lines: 12
Contained by: module functions_global
Modules used: galacticus_error

subroutine: `tasks_evolve_forest_construct_null`
Code lines: 14
Contained by: module functions_global
Modules used: galacticus_error input_parameters

subroutine: `tasks_evolve_forest_destruct_null`
Code lines: 10
Contained by: module functions_global
Modules used: galacticus_error

subroutine: `tasks_evolve_forest_perform_null`
Code lines: 13
Contained by: module functions_global
Modules used: galacticus_error

function: `virial_density_contrast_percolation_objects_constructor_null`
Code lines: 13
Contained by: module functions_global
Modules used: galacticus_error input_parameters

function: `virial_density_contrast_percolation_solver_null`
Code lines: 25
Contained by: module functions_global
Modules used: galacticus_error

file: `functions.global.utilities.F90`
Description: Contains a module which provides utilities needed by global functions.
Code lines: 74

module: `functions_global_utilities`
Description: Provides utilities needed by global functions.
Code lines: 51
Contained by: file functions.global.utilities.F90
Used by: program galacticus program test_dark_matter_profiles_-generic

subroutine: `functions_global_set`
Description: Set pointers to all global functions.
Code lines: 42
Contained by: module functions_global_utilities
Modules used: functions_global galacticus_calculations_resets galacticus_state tasks_evolve_forests_utilities

virial_density_contrast_percolation_-
utilities

file: galactic.filters.F90
Description: Contains a module which provides a class that implements galactic filters.
Code lines: 45

module: galactic_filters
Description: Provides an object that implements galactic filters.
Code lines: 23
Contained by: file galactic.filters.F90
Modules used: galacticus_nodes
Used by: file dark_matter_-profiles.structure.scale.binary.F90; function localgroupmassfunctionconstructorinternal; file galacticus.output.analyses.mean_-function_1d.F90; file galacticus.output.analyses.volume_-function_1d.F90; subroutine galacticus_state_store; file tasks.evolve_forests.F90; subroutine galacticus_function_-classes_destroy; file galacticus.output.analyses.correlation_-function.F90; file galacticus.output.analyses.scatter_-function_1d.F90; subroutine galacticus_state_retrieve; file merger_-trees.outputter.standard.F90

type: filterlist
Code lines: 3
Contained by: module galactic_filters

file: galactic.filters.ISM_mass.F90
Description: Contains a module which implements a galactic high-pass filter for total ISM mass.
Code lines: 88

interface: galacticfilterismmass
Description: Constructors for the "ismMass" galactic filter class.
Code lines: 4
Contained by: file galactic.filters.ISM_mass.F90

function: ismmassconstructorinternal
Description: Internal constructor for the "ismMass" galactic filter class.
Code lines: 7
Contained by: file galactic.filters.ISM_mass.F90

function: ismmassconstructorparameters
Description: Constructor for the "ismMass" galactic filter class which takes a parameter set as input.
Code lines: 18
Contained by: file galactic.filters.ISM_mass.F90

Modules used: input_parameters

function: ismmasspasses
Description: Implement an ismMass-pass galactic filter.
Code lines: 15
Contained by: file galactic.filters.ISM_mass.F90
Modules used: galacticus_nodes

file: galactic.filters.all.F90
Description: Contains a module which implements a galactic filter class which is the "all" combination of a set of other filters.
Code lines: 155

function: allconstructorinternal
Description: Internal constructor for the "all" filter class.
Code lines: 14
Contained by: file galactic.filters.all.F90

function: allconstructorparameters
Description: Constructor for the "all" galactic filter class which takes a parameter set as input.
Code lines: 22
Contained by: file galactic.filters.all.F90
Modules used: input_parameters

subroutine: alldeepcopy
Description: Perform a deep copy for the all galactic filter class.
Code lines: 31
Contained by: file galactic.filters.all.F90
Modules used: galacticus_error

subroutine: alldestructor
Description: Destructor for the all galactic filter class.
Code lines: 16
Contained by: file galactic.filters.all.F90

function: allpasses
Description: Apply a set of filters to a node combined with "all" operations.
Code lines: 20
Contained by: file galactic.filters.all.F90

interface: galacticfilterall
Description: Constructors for the all galactic filter class.
Code lines: 4
Contained by: file galactic.filters.all.F90

file: galactic.filters.always.F90
Description: Contains a module which implements a galactic filter which always passes.
Code lines: 60

function: `alwaysconstructorparameters`
Description: Constructor for the "always" galactic filter class which takes a parameter set as input.
Code lines: 10
Contained by: file `galactic.filters.always.F90`
Modules used: `input_parameters`

function: `alwayspasses`
Description: Implement an always-pass galactic filter.
Code lines: 9
Contained by: file `galactic.filters.always.F90`

interface: `galacticfilteralways`
Description: Constructors for the "always" galactic filter class.
Code lines: 3
Contained by: file `galactic.filters.always.F90`

file: `galactic.filters.any.F90`
Description: Contains a module which implements a galactic filter class which is the "any" combination of a set of other filters.
Code lines: 155

function: `anyconstructorinternal`
Description: Internal constructor for the "any" filter class.
Code lines: 14
Contained by: file `galactic.filters.any.F90`

function: `anyconstructorparameters`
Description: Constructor for the "any" galactic filter class which takes a parameter set as input.
Code lines: 22
Contained by: file `galactic.filters.any.F90`
Modules used: `input_parameters`

subroutine: `anydeepcopy`
Description: Perform a deep copy for the `any` galactic filter class.
Code lines: 31
Contained by: file `galactic.filters.any.F90`
Modules used: `galacticus_error`

subroutine: `anydestructor`
Description: Destructor for the "any" galactic filter class.
Code lines: 16
Contained by: file `galactic.filters.any.F90`

function: `anypasses`
Description: Apply a set of filters to a `node` combined with "any" operations.
Code lines: 20
Contained by: file `galactic.filters.any.F90`

interface: `galacticfilterany`
Description: Constructors for the any galactic filter class.
Code lines: 4
Contained by: file `galactic.filters.any.F90`

file: `galactic.filters.basic_mass.F90`
Description: Contains a module which implements a galactic high-pass filter for the default "basic" halo mass.
Code lines: 84

function: `basicmassconstructorinternal`
Description: Internal constructor for the "basicMass" galactic filter class.
Code lines: 7
Contained by: file `galactic.filters.basic_mass.F90`

function: `basicmassconstructorparameters`
Description: Constructor for the "basicMass" galactic filter class which takes a parameter set as input.
Code lines: 18
Contained by: file `galactic.filters.basic_mass.F90`
Modules used: `input_parameters`

function: `basicmasspasses`
Description: Implement a basic mass high-pass galactic filter.
Code lines: 11
Contained by: file `galactic.filters.basic_mass.F90`
Modules used: `galacticus_nodes`

interface: `galacticfilterbasicmass`
Description: Constructors for the "basicMass" galactic filter class.
Code lines: 4
Contained by: file `galactic.filters.basic_mass.F90`

file: `galactic.filters.formation_time.F90`
Description: Contains a module which implements a galactic filter which removes recently-formed halos.
Code lines: 89

function: `formationtimeconstructorinternal`
Description: Internal constructor for the "formationTime" galactic filter class.
Code lines: 8
Contained by: file `galactic.filters.formation_time.F90`

function: `formationtimeconstructorparameters`
Description: Constructor for the "formationTime" galactic filter class which takes a parameter set as input.
Code lines: 20
Contained by: file `galactic.filters.formation_time.F90`
Modules used: `input_parameters`

function: `formationtimepasses`

Description: Implement a filter which rejects halos that formed too recently.
Code lines: 13
Contained by: file galactic.filters.formation_time.F90
Modules used: galacticus_nodes

interface: galacticfilterformationtime
Description: Constructors for the "formationTime" galactic filter class.
Code lines: 4
Contained by: file galactic.filters.formation_time.F90

file: galactic.filters.halo_isolated.F90
Description: Contains a module which implements a filter which passes only isolated halos.
Code lines: 60

interface: galacticfilterhaloisolated
Description: Constructors for the "haloIsolated" galactic filter class.
Code lines: 3
Contained by: file galactic.filters.halo_isolated.F90

function: haloisolatedconstructorparameters
Description: Constructor for the "haloIsolated" galactic filter class which takes a parameter set as input.
Code lines: 10
Contained by: file galactic.filters.halo_isolated.F90
Modules used: input_parameters

function: haloisolatedpasses
Description: Implement a galactic filter which passes only isolated halos.
Code lines: 9
Contained by: file galactic.filters.halo_isolated.F90

file: galactic.filters.halo_mass.F90
Description: Contains a module which implements a galactic high-pass filter for halo mass under a given definition.
Code lines: 104
Modules used: virial_density_contrast

interface: galacticfilterhalomass
Description: Constructors for the "haloMass" galactic filter class.
Code lines: 4
Contained by: file galactic.filters.halo_mass.F90

function: halomassconstructorinternal
Description: Internal constructor for the "haloMass" galactic filter class.
Code lines: 9
Contained by: file galactic.filters.halo_mass.F90

function: halomassconstructorparameters
Description: Constructor for the "haloMass" galactic filter class which takes a parameter set as input.
Code lines: 22

Contained by: file galactic.filters.halo_mass.F90
Modules used: input_parameters

subroutine: halomassdestructor
Description: Destructor for the "haloMass" galactic filter class.
Code lines: 7
Contained by: file galactic.filters.halo_mass.F90

function: halomasspasses
Description: Implement a halo mass high-pass galactic filter.
Code lines: 12
Contained by: file galactic.filters.halo_mass.F90
Modules used: dark_matter_profile_mass_definitions galacticus_nodes

file: galactic.filters.halo_not_isolated.F90
Description: Contains a module which implements a filter which passes only non-isolated halos.
Code lines: 60

interface: galacticfilterhalonotisolated
Description: Constructors for the "haloNotIsolated" galactic filter class.
Code lines: 3
Contained by: file galactic.filters.halo_not_isolated.F90

function: halonotisolatedconstructorparameters
Description: Constructor for the "haloNotIsolated" galactic filter class which takes a parameter set as input.
Code lines: 10
Contained by: file galactic.filters.halo_not_isolated.F90
Modules used: input_parameters

function: halonotisolatedpasses
Description: Implement a galactic filter which passes only isolated halos.
Code lines: 9
Contained by: file galactic.filters.halo_not_isolated.F90

file: galactic.filters.host_mass_range.F90
Description: Implements a galactic filter which passes nodes with host halo basic mass within a specified range.
Code lines: 92

interface: galacticfilterhostmassrange
Description: Constructors for the "hostMassRange" galactic filter class.
Code lines: 4
Contained by: file galactic.filters.host_mass_range.F90

function: hostmassrangeconstructorinternal
Description: Internal constructor for the "hostMassRange" galactic filter class.
Code lines: 7
Contained by: file galactic.filters.host_mass_range.F90

function: `hostmassrangeconstructorparameters`
Description: Constructor for the "hostMassRange" galactic filter class which takes a parameter set as input.
Code lines: 25
Contained by: file `galactic.filters.host_mass_range.F90`
Modules used: `input_parameters`

function: `hostmassrangepasses`
Description: Implement a filter which passes nodes with host halo basic mass in a specified range.
Code lines: 15
Contained by: file `galactic.filters.host_mass_range.F90`
Modules used: `galacticus_nodes`

file: `galactic.filters.lightcone.F90`
Description: Contains a module which implements a galactic filter on lightcone geometry.
Code lines: 88
Modules used: `geometry_lightcones`

interface: `galacticfilterlightcone`
Description: Constructors for the "lightcone" galactic filter class.
Code lines: 4
Contained by: file `galactic.filters.lightcone.F90`

function: `lightconeconstructorinternal`
Description: Internal constructor for the "lightcone" galactic filter class.
Code lines: 8
Contained by: file `galactic.filters.lightcone.F90`

function: `lightconeconstructorparameters`
Description: Constructor for the "lightcone" galactic filter class which takes a parameter set as input.
Code lines: 13
Contained by: file `galactic.filters.lightcone.F90`
Modules used: `input_parameters`

subroutine: `lightconedestructor`
Description: Destructor for the `lightcone` galactic filter class.
Code lines: 7
Contained by: file `galactic.filters.lightcone.F90`

function: `lightconepasses`
Description: Implement a lightcone geometry galactic filter.
Code lines: 8
Contained by: file `galactic.filters.lightcone.F90`

file: `galactic.filters.main_branch.F90`
Description: Contains a module which implements a filter which passes only main branch halos.
Code lines: 60

interface: `galacticfiltermainbranch`
Description: Constructors for the "mainBranch" galactic filter class.
Code lines: 3
Contained by: file `galactic.filters.main_branch.F90`

function: `mainbranchconstructorparameters`
Description: Constructor for the "mainBranch" galactic filter class which takes a parameter set as input.
Code lines: 10
Contained by: file `galactic.filters.main_branch.F90`
Modules used: `input_parameters`

function: `mainbranchpasses`
Description: Implement a galactic filter which passes only main branch halos.
Code lines: 9
Contained by: file `galactic.filters.main_branch.F90`

file: `galactic.filters.major_node_merger.recent.F90`
Description: Contains a module which implements a galactic low-pass filter for time since the last major node merger.
Code lines: 88

interface: `galacticfilternodemajormergerrecent`
Description: Constructors for the "nodeMajorMergerRecent" galactic filter class.
Code lines: 4
Contained by: file `galactic.filters.major_node_merger.recent.F90`

function: `nodemajormergerrecentconstructorinternal`
Description: Internal constructor for the "nodeMajorMergerRecent" galactic filter class.
Code lines: 7
Contained by: file `galactic.filters.major_node_merger.recent.F90`

function: `nodemajormergerrecentconstructorparameters`
Description: Constructor for the "nodeMajorMergerRecent" galactic filter class which takes a parameter set as input.
Code lines: 20
Contained by: file `galactic.filters.major_node_merger.recent.F90`
Modules used: `input_parameters`

function: `nodemajormergerrecentpasses`
Description: Implement a low-pass filter for time since the last major node merger.
Code lines: 13
Contained by: file `galactic.filters.major_node_merger.recent.F90`
Modules used: `galacticus_nodes`

file: `galactic.filters.not.F90`
Description: Contains a module which implements an inverting filter.
Code lines: 84

interface: `galacticfilternot`
Description: Constructors for the "not" galactic filter class.
Code lines: 4
Contained by: file `galactic.filters.not.F90`

function: `notconstructorinternal`
Description: Internal constructor for the "not" galactic filter class.
Code lines: 8
Contained by: file `galactic.filters.not.F90`

function: `notconstructorparameters`
Description: Constructor for the "not" galactic filter class which takes a parameter set as input.
Code lines: 13
Contained by: file `galactic.filters.not.F90`
Modules used: `input_parameters`

subroutine: `notdestructor`
Description: Destructor for the `not` galactic filter class.
Code lines: 7
Contained by: file `galactic.filters.not.F90`

function: `notpasses`
Description: Implement a not galactic filter.
Code lines: 8
Contained by: file `galactic.filters.not.F90`

file: `galactic.filters.root_node.F90`
Description: Contains a module which implements a filter which passes only nodes that are roots of their merger tree.
Code lines: 60

interface: `galacticfilterrootnode`
Description: Constructors for the "rootNode" galactic filter class.
Code lines: 3
Contained by: file `galactic.filters.root_node.F90`

function: `rootnodeconstructorparameters`
Description: Constructor for the "rootNode" galactic filter class which takes a parameter set as input.
Code lines: 10
Contained by: file `galactic.filters.root_node.F90`
Modules used: `input_parameters`

function: `rootnodepasses`
Description: Implement a galactic filter which passes only main branch halos.
Code lines: 9
Contained by: file `galactic.filters.root_node.F90`

file: `galactic.filters.spheroid_stellar_mass.F90`

Description: Contains a module which implements a galactic high-pass filter for spheroid stellar mass.
Code lines: 88

interface: `galacticfilterspheroidstellarmass`
Description: Constructors for the "spheroidStellarMass" galactic filter class.
Code lines: 4
Contained by: file `galactic.filters.spheroid_stellar_mass.F90`

function: `spheroidstellarmassconstructorinternal`
Description: Internal constructor for the "spheroidStellarMass" galactic filter class.
Code lines: 8
Contained by: file `galactic.filters.spheroid_stellar_mass.F90`

function: `spheroidstellarmassconstructorparameters`
Description: Constructor for the "spheroidStellarMass" galactic filter class which takes a parameter set as input.
Code lines: 19
Contained by: file `galactic.filters.spheroid_stellar_mass.F90`
Modules used: `input_parameters`

function: `spheroidstellarmasspasses`
Description: Implement a stellar mass high-pass galactic filter.
Code lines: 13
Contained by: file `galactic.filters.spheroid_stellar_mass.F90`
Modules used: `galacticus_nodes`

file: `galactic.filters.star_formation_rate.F90`
Description: Contains a module which implements a galactic high-pass filter for total star formation rate.
Code lines: 122

interface: `galacticfilterstarformationrate`
Description: Constructors for the "starFormationRate" galactic filter class.
Code lines: 4
Contained by: file `galactic.filters.star_formation_rate.F90`

function: `starformationrateconstructorinternal`
Description: Internal constructor for the "starFormationRate" galactic filter class.
Code lines: 10
Contained by: file `galactic.filters.star_formation_rate.F90`

function: `starformationrateconstructorparameters`
Description: Constructor for the "starFormationRate" galactic filter class which takes a parameter set as input.
Code lines: 37
Contained by: file `galactic.filters.star_formation_rate.F90`
Modules used: `input_parameters`

function: `starformationratepasses`
Description: Implement an starFormationRate-pass galactic filter.

Code lines: 21
Contained by: file galactic.filters.star_formation_rate.F90
Modules used: galacticus_nodes

file: galactic.filters.stellar_absolute_magnitudes.F90
Description: Contains a module which implements a galactic low-pass (i.e. bright-pass) filter for stellar absolute magnitudes.
Code lines: 125

interface: galacticfilterstellarabsolutemagnitudes
Description: Constructors for the "stellarAbsoluteMagnitudes" galactic filter class.
Code lines: 4
Contained by: file galactic.filters.stellar_absolute_magnitudes.F90

function: stellarabsolutemagnitudesconstructorinternal
Description: Internal constructor for the "stellarAbsoluteMagnitudes" galactic filter class.
Code lines: 13
Contained by: file galactic.filters.stellar_absolute_magnitudes.F90
Modules used: galacticus_error stellar_luminosities_structure

function: stellarabsolutemagnitudesconstructorparameters
Description: Constructor for the "stellarAbsoluteMagnitudes" galactic filter class which takes a parameter set as input.
Code lines: 24
Contained by: file galactic.filters.stellar_absolute_magnitudes.F90
Modules used: galacticus_error input_parameters
memory_management stellar_luminosities_structure

function: stellarabsolutemagnitudespasses
Description: Implement a stellar absolute magnitude low-pass galactic filter.
Code lines: 40
Contained by: file galactic.filters.stellar_absolute_magnitudes.F90
Modules used: galactic_structure_enclosed_masses galactic_structure_options
galacticus_nodes stellar_luminosities_structure

file: galactic.filters.stellar_apparent_magnitudes.F90
Description: Contains a module which implements a galactic low-pass (i.e. bright-pass) filter for stellar apparent magnitudes.
Code lines: 152
Modules used: cosmology_functions

interface: galacticfilterstellarapparentmagnitudes
Description: Constructors for the "stellarApparentMagnitudes" galactic filter class.
Code lines: 4
Contained by: file galactic.filters.stellar_apparent_magnitudes.F90

function: stellarapparentmagnitudesconstructorinternal
Description: Internal constructor for the "stellarApparentMagnitudes" galactic filter class.
Code lines: 14

Contained by: file galactic.filters.stellar_apparent_magnitudes.F90
Modules used: galacticus_error stellar_luminosities_structure

function: stellarapparentmagnitudesconstructorparameters
Description: Constructor for the "stellarApparentMagnitudes" galactic filter class which takes a parameter set as input.
Code lines: 27
Contained by: file galactic.filters.stellar_apparent_magnitudes.F90
Modules used: galacticus_error input_parameters memory_management stellar_luminosities_structure

subroutine: stellarapparentmagnitudesdestructor
Description: Destructor for the "stellarApparentMagnitudes" galactic filter class.
Code lines: 7
Contained by: file galactic.filters.stellar_apparent_magnitudes.F90

function: stellarapparentmagnitudespasses
Description: Implement a stellar apparent magnitude low-pass galactic filter.
Code lines: 50
Contained by: file galactic.filters.stellar_apparent_magnitudes.F90
Modules used: galactic_structure_enclosed_masses galactic_structure_options galacticus_nodes stellar_luminosities_structure

file: galactic.filters.stellar_mass.F90
Description: Contains a module which implements a galactic high-pass filter for total stellar mass.
Code lines: 88

interface: galacticfilterstellarmass
Description: Constructors for the "stellarMass" galactic filter class.
Code lines: 4
Contained by: file galactic.filters.stellar_mass.F90

function: stellarmassconstructorinternal
Description: Internal constructor for the "stellarMass" galactic filter class.
Code lines: 7
Contained by: file galactic.filters.stellar_mass.F90

function: stellarmassconstructorparameters
Description: Constructor for the "stellarMass" galactic filter class which takes a parameter set as input.
Code lines: 18
Contained by: file galactic.filters.stellar_mass.F90
Modules used: input_parameters

function: stellarmasspasses
Description: Implement a stellar mass high-pass galactic filter.
Code lines: 15
Contained by: file galactic.filters.stellar_mass.F90
Modules used: galacticus_nodes

file: `galactic.filters.stellar_mass_morphology.F90`
Description: Contains a module which implements a galactic high-pass filter for stellar mass-weighted morphology (i.e. spheroid-to-total ratio).
Code lines: 93

interface: `galacticfilterstellarmassmorphology`
Description: Constructors for the "stellarMassMorphology" galactic filter class.
Code lines: 4
Contained by: file `galactic.filters.stellar_mass_morphology.F90`

function: `stellarmassmorphologyconstructorinternal`
Description: Internal constructor for the "stellarMassMorphology" galactic filter class.
Code lines: 7
Contained by: file `galactic.filters.stellar_mass_morphology.F90`

function: `stellarmassmorphologyconstructorparameters`
Description: Constructor for the "stellarMassMorphology" galactic filter class which takes a parameter set as input.
Code lines: 18
Contained by: file `galactic.filters.stellar_mass_morphology.F90`
Modules used: `input_parameters`

function: `stellarmassmorphologypasses`
Description: Implement a stellar mass-weighted morphology high-pass galactic filter.
Code lines: 19
Contained by: file `galactic.filters.stellar_mass_morphology.F90`
Modules used: `galacticus_nodes`

file: `galactic.filters.survey_geometry.F90`
Description: Contains a module which implements a filter which passes only nodes that lie within a survey geometry.
Code lines: 96
Modules used: `geometry_surveys`

interface: `galacticfiltersurveygeometry`
Description: Constructors for the "surveyGeometry" galactic filter class.
Code lines: 4
Contained by: file `galactic.filters.survey_geometry.F90`

function: `surveygeometryconstructorinternal`
Description: Internal constructor for the "surveyGeometry" galactic filter class.
Code lines: 8
Contained by: file `galactic.filters.survey_geometry.F90`

function: `surveygeometryconstructorparameters`
Description: Constructor for the "surveyGeometry" galactic filter class which takes a parameter set as input.
Code lines: 14
Contained by: file `galactic.filters.survey_geometry.F90`

Modules used: geometry_surveys input_parameters

subroutine: surveygeometrydestructor
Description: Destructor for the "surveyGeometry" galactic filter class.
Code lines: 7
Contained by: file galactic.filters.survey_geometry.F90

function: surveygeometrypasses
Description: Implement a galactic filter which passes only nodes with a survey geometry.
Code lines: 17
Contained by: file galactic.filters.survey_geometry.F90
Modules used: galacticus_nodes

file: galactic.filters.tree_hosted.F90
Description: Contains a module which implements a filter which passes only nodes that are hosted in a merger tree.
Code lines: 60

interface: galacticfiltertreehosted
Description: Constructors for the "treeHosted" galactic filter class.
Code lines: 3
Contained by: file galactic.filters.tree_hosted.F90

function: treehostedconstructorparameters
Description: Constructor for the "treeHosted" galactic filter class which takes a parameter set as input.
Code lines: 10
Contained by: file galactic.filters.tree_hosted.F90
Modules used: input_parameters

function: treehostedpasses
Description: Implement a galactic filter which passes only main branch halos.
Code lines: 9
Contained by: file galactic.filters.tree_hosted.F90

file: galactic_dynamics.bar_instability.Efstathiou1982.F90
Description: Implementation of the Efstathiou et al. [1982] model for galactic disk bar instability.
Code lines: 180

function: efstathiou1982constructorinternal
Description: Internal constructor for the efstathiou1982 model for galactic disk bar instability class.
Code lines: 8
Contained by: file galactic_dynamics.bar_instability.Efstathiou1982.F90

function: efstathiou1982constructorparameters
Description: Constructor for the efstathiou1982 model for galactic disk bar instability class which takes a parameter set as input.
Code lines: 36
Contained by: file galactic_dynamics.bar_instability.Efstathiou1982.F90
Modules used: input_parameters

function: `efstathiou1982estimator`
Description: Compute the stability estimator for the Efstathiou et al. [1982] model for galactic disk bar instability.
Code lines: 26
Contained by: file `galactic_dynamics.bar_instability.Efstathiou1982.F90`
Modules used: `galacticus_nodes` `numerical_constants_astronomical`

subroutine: `efstathiou1982timescale`
Description: Computes a timescale for depletion of a disk to a pseudo-bulge via bar instability based on the criterion of Efstathiou et al. [1982].
Code lines: 50
Contained by: file `galactic_dynamics.bar_instability.Efstathiou1982.F90`
Modules used: `galacticus_nodes` `numerical_constants_astronomical` `numerical_constants_physical` `numerical_constants_prefixes`

interface: `galacticdynamicsbarinstabilityefstathiou1982`
Description: Constructors for the `efstathiou1982` model for galactic disk bar instability class.
Code lines: 4
Contained by: file `galactic_dynamics.bar_instability.Efstathiou1982.F90`

file: `galactic_dynamics.bar_instability.Efstathiou1982Tidal.F90`
Description: Implementation of the Efstathiou et al. [1982] model for galactic disk bar instability, but including the effects of tidal forces.
Code lines: 162
Modules used: `satellites_tidal_fields`

subroutine: `efstathiou1982destructor`
Description: Destructor for the `efstathiou1982` model for galactic disk bar instability class.
Code lines: 7
Contained by: file `galactic_dynamics.bar_instability.Efstathiou1982Tidal.F90`

function: `efstathiou1982tidalconstructorinternal`
Description: Internal constructor for the `efstathiou1982Tidal` model for galactic disk bar instability class.
Code lines: 10
Contained by: file `galactic_dynamics.bar_instability.Efstathiou1982Tidal.F90`

function: `efstathiou1982tidalconstructorparameters`
Description: Constructor for the `efstathiou1982Tidal` model for galactic disk bar instability class which takes a parameter set as input.
Code lines: 21
Contained by: file `galactic_dynamics.bar_instability.Efstathiou1982Tidal.F90`
Modules used: `input_parameters`

function: `efstathiou1982tidalestimator`
Description: Compute the stability estimator for the Efstathiou et al. [1982] model for galactic disk bar instability.
Code lines: 25
Contained by: file `galactic_dynamics.bar_instability.Efstathiou1982Tidal.F90`
Modules used: `galacticus_nodes` `numerical_constants_astronomical`

function: `efstathiou1982tidaltidaltensorradial`
Description: Compute the radial part of the tidal tensor.
Code lines: 15
Contained by: file `galactic_dynamics.bar_instability.Efstathiou1982Tidal.F90`
Modules used: `galacticus_nodes`

subroutine: `efstathiou1982tidaltimescale`
Description: Computes a timescale for depletion of a disk to a pseudo-bulge via bar instability based on the criterion of Efstathiou et al. [1982].
Code lines: 17
Contained by: file `galactic_dynamics.bar_instability.Efstathiou1982Tidal.F90`
Modules used: `galacticus_nodes`

interface: `galacticdynamicsbarinstabilityefstathiou1982tidal`
Description: Constructors for the `efstathiou1982Tidal` model for galactic disk bar instability class.
Code lines: 4
Contained by: file `galactic_dynamics.bar_instability.Efstathiou1982Tidal.F90`

file: `galactic_dynamics.bar_instability.F90`
Description: Contains a module which provides a class that implements calculations of bar instability in galactic disks.
Code lines: 41

module: `galactic_dynamics_bar_instabilities`
Description: Provides a class that implements calculations of bar instability in galactic disks.
Code lines: 19
Contained by: file `galactic_dynamics.bar_instability.F90`
Modules used: `galacticus_nodes`
Used by: subroutine `galacticus_function_classes_destroy`; subroutine `galacticus_state_retrieve`; subroutine `galacticus_state_store`; module `node_component_disk_standard`; module `node_component_dynamics_statistics_bars`

file: `galactic_dynamics.bar_instability.fixed_timescale.F90`
Description: Implementation of a simple model for galactic disk bar instability in which the timescale is fixed.
Code lines: 84

function: `fixedtimescaleconstructorinternal`
Description: Internal constructor for the `fixedTimescale` model for galactic disk bar instability class.
Code lines: 8
Contained by: file `galactic_dynamics.bar_instability.fixed_timescale.F90`

function: `fixedtimescaleconstructorparameters`
Description: Constructor for the `fixedTimescale` model for galactic disk bar instability class which takes a parameter set as input.
Code lines: 20
Contained by: file `galactic_dynamics.bar_instability.fixed_timescale.F90`

Modules used: input_parameters

subroutine: fixedtimescaletimescale
Description: Assume a constant timescale for depletion of a disk to a pseudo-bulge via bar instability.
Code lines: 11
Contained by: file galactic_dynamics.bar_instability.fixed_timescale.F90

interface: galacticdynamicsbarinstabilityfixedtimescale
Description: Constructors for the fixedTimescale model for galactic disk bar instability class.
Code lines: 4
Contained by: file galactic_dynamics.bar_instability.fixed_timescale.F90

file: galactic_dynamics.bar_instability.stable.F90
Description: Implementation of a perfectly stable model for galactic disk bar instability.
Code lines: 68

interface: galacticdynamicsbarinstabilitystable
Description: Constructors for the stable model for galactic disk bar instability class.
Code lines: 3
Contained by: file galactic_dynamics.bar_instability.stable.F90

function: stableconstructorparameters
Description: Constructor for the stable model for galactic disk bar instability class which takes a parameter set as input.
Code lines: 11
Contained by: file galactic_dynamics.bar_instability.stable.F90
Modules used: input_parameters

subroutine: stabletimescale
Description: Computes a timescale for depletion of a disk to a pseudo-bulge via bar instability based on the criterion of Efstathiou et al. [1982].
Code lines: 16
Contained by: file galactic_dynamics.bar_instability.stable.F90
Modules used: numerical_constants_astronomical numerical_constants_physical

file: galactic_structure.density.F90
Description: Contains a module which implements calculations of the density at a specific position.
Code lines: 139

module: galactic_structure_densities
Description: Implements calculations of the density at a specific position.
Code lines: 117
Contained by: file galactic_structure.density.F90
Used by: function standardgrowthrate
function galactic_structure_velocity_dispersion
function velocity_dispersion_integrand
function densityprofileextract
function projecteddensityextract
function projecteddensityintegrand
function velocitydispersiondensityintegrand
function velocitydispersionsurfacedensityintegrand

function velocitydispersionvelocitydensityintegrand
function velocitydispersionvelocitysurfacedensityintegrand
subroutine node_component_black_hole_-standard_mass_accretion_rate
subroutine node_component_hot_halo_-cold_mode_outflow_return
subroutine node_component_hot_halo_-cold_mode_rate_compute
subroutine node_component_satellite_-orbiting_rate_compute
function simpleratemassloss
function chandrasekhar1943acceleration
function sphericalsymmetrytidaltensorradial
function gnedin1999heatingrate
function zentner2005masslossrate

function: `component_density`

Description: Unary function returning the density in a component. Suitable for mapping over components.
Code lines: 8
Contained by: module galactic_structure_densities
Modules used: galacticus_nodes

function: `galactic_structure_density`

Description: Compute the density (of given `massType`) at the specified `position`. Assumes that galactic structure has already been computed.
Code lines: 93
Contained by: module galactic_structure_densities
Modules used: coordinate_systems, dark_matter_profile_structure_tasks, galactic_structure_options, galacticus_error, galacticus_nodes, hot_halo_mass_distributions, node_component_hot_halo_cold_mode_-structure_tasks

file: `galactic_structure.enclosed_mass.F90`

Description: Contains a module which implements calculations of the mass enclosed within a specified radius.
Code lines: 287

module: `galactic_structure_enclosed_masses`

Description: Implements calculations of the mass enclosed within a specified radius.
Code lines: 265
Contained by: file galactic_structure.enclosed_mass.F90
Modules used: galactic_structure_options, galacticus_nodes
Used by: function tidalradiusroot, function tidalradiusseparationinitial, function stellarabsolutemagnitudespasses, function stellarapparentmagnitudespasses, function galactic_structure_velocity_-dispersion, function velocity_dispersion_integrand, function font2008radiussolver, function massenclosed, subroutine historystore, subroutine recordevolutionstore, subroutine particulateoperate, function densityprofileextract, function densitycontrastsextract, function densitycontrastsroot

function radiihalflightpropertiesextract
function radiushalfmassextract
function luminositystellarextract
function massismextract
function massblackholeextract
function massprofileextract
function massstellarextract
function massstellarmorphologyextract
function massstellarspheroidextract
function projecteddensityextract
function halfmassradiusextract
function rotationcurveextract
function velocitydispersionextract
function velocitydispersionvelocitydensityintegrand
subroutine node_component_hot_halo_cold_mode_node_merger
subroutine node_component_hot_halo_standard_node_merger
subroutine node_component_satellite_orbiting_bound_mass_initialize
subroutine node_component_satellite_orbiting_rate_compute
subroutine node_component_satellite_orbiting_scale_set
function simpleratemassloss
subroutine baugh2005get
subroutine simpleget
subroutine verysimpleget
subroutine cole2000get
function cole2000halfmassradiusroot
subroutine simpleget
subroutine standardget
subroutine cole2000get
function tracedarkmatterinversecmfradial
function sphericalsymmetrytidaltensorradial
function gnedin1999heatingrate
function zentner2005masslossrate
function zentner2005tidalradiussolver
subroutine halomodelgenerateperform

function: `component_enclosed_mass`

Description: Unary function returning the enclosed mass in a component. Suitable for mapping over components.
Code lines: 8
Contained by: module galactic_structure_enclosed_masses
Modules used: galacticus_nodes

function: `enclosed_density_root`

Description: Root function used in solving for the radius that encloses a given density.
Code lines: 7
Contained by: module galactic_structure_enclosed_masses
Modules used: numerical_constants_math

function: `enclosed_mass_root`

Description: Root function used in solving for the radius that encloses a given mass.
Code lines: 7
Contained by: module galactic_structure_enclosed_masses

function: `galactic_structure_enclosed_mass`

Description: Solve for the mass within a given radius, or the total mass if no radius is specified. Assumes that galactic structure has already been computed.
Code lines: 58
Contained by: module galactic_structure_enclosed_masses
Modules used: dark_matter_profile_structure_tasks, galacticus_nodes, hot_halo_mass_distributions, node_component_hot_halo_cold_mode_structure_tasks

subroutine: galactic_structure_enclosed_mass_defaults

Description: Set the default values for options in the enclosed mass functions.
Code lines: 30
Contained by: module galactic_structure_enclosed_masses
Modules used: galacticus_error

function: galactic_structure_radius_enclosing_density

Description: Return the radius enclosing a given density (or density contrast) in thisNode.
Code lines: 49
Contained by: module galactic_structure_enclosed_masses
Modules used: cosmology_functions, dark_matter_halo_scales, galacticus_display, galacticus_error, galacticus_nodes, iso_varying_string, root_finder, string_handling

function: galactic_structure_radius_enclosing_mass

Description: Return the radius enclosing a given mass (or fractional mass) in thisNode.
Code lines: 76
Contained by: module galactic_structure_enclosed_masses
Modules used: dark_matter_halo_scales, dark_matter_profiles, galacticus_display, galacticus_error, iso_varying_string, kind_numbers, root_finder, string_handling

file: galactic_structure.options.F90

Description: Contains a module which provides various enumerations for the galactic structure functions.
Code lines: 85

module: galactic_structure_options

Description: Provides various internal option codes for the galactic structure functions.
Code lines: 63
Contained by: file galactic_structure.options.F90
Used by: function tidalradiusroot, function tidalradiusseparationinitial, function standardgrowthrate, file dark_matter_profiles.adiabatic_-Gnedin2004.F90, function genericpotentialdifferencenumerical, function genericpotentialnumerical, function dark_matter_profile_density_-task, function dark_matter_profile_enclosed_-mass_task, function dark_matter_profile_-potential_task, function dark_matter_profile_rotation_-curve_gradient_task, function dark_matter_profile_rotation_-curve_task, function burkertpotential, function einastopotential, function nfwpotential, function heatedradiusenclosingmass, function isothermalpotential, function stellarabsolutemagnitudespasses, function stellarapparentmagnitudespasses

function galactic_structure_density
module galactic_structure_enclosed_masses
module galactic_structure_potentials
subroutine galactic_structure_radii_definition_decode
subroutine radiussolve
module galactic_structure_rotation_curves
module galactic_structure_rotation_curve_gradients
module galactic_structure_surface_densities
function galactic_structure_velocity_dispersion
function hothalomassdistributiondensity
function hothalomassdistributionenclosedmass
function hothalomassdistributionrotationcurve
function hothalomassdistributionrotationcurvegradient
function font2008radiussolver
function massenclosed
subroutine historystore
subroutine recordevolutionstore
subroutine particulateoperate
function sedfitevaluate
function densityprofileextract
function densitycontrastsconstructorinternal
function densitycontrastsextract
function densitycontrastsroot
function radiihalflightpropertiesextract
function radiushalfmassextract
function luminositystellarextract
function massismextract
function massblackholeextract
function massprofileconstructorinternal
function massprofileextract
function massstellarextract
function massstellarmorphologyextract
function massstellarspheroidextract
function projecteddensityextract
function halfmassradiusextract
function rotationcurveextract
function velocitydispersionextract
function velocitydispersionlambdarintegrand1
function velocitydispersionlambdarintegrand2
function velocitydispersionvelocitydensityintegrand
function node_component_black_hole_simple_enclosed_mass
function node_component_black_hole_standard_enclosed_mass
function node_component_disk_standard_density
function node_component_disk_standard_enclosed_mass
function node_component_disk_standard_potential
function node_component_disk_standard_rotation_curve
function node_component_disk_standard_rotation_curve_gradient
function node_component_disk_standard_surface_density
function node_component_disk_very_simple_enclosed_mass
function node_component_spheroid_standard_density
function node_component_spheroid_standard_enclosed_mass
function node_component_spheroid_standard_potential
function node_component_spheroid_standard_rotation_curve
function node_component_spheroid_standard_rotation_curve_gradient

function node_component_spheroid_very_simple_enclosed_mass
function node_component_black_hole_simple_potential
function node_component_black_hole_simple_rotation_curve_gradient
function node_component_black_hole_standard_recoil_escapes
function node_component_black_hole_standard_rotation_curve
subroutine node_component_disk_standard_rate_compute
subroutine node_component_disk_very_simple_rates
subroutine node_component_hot_halo_cold_mode_outflow_return
function node_component_hot_halo_cold_mode_density_task
function node_component_hot_halo_cold_mode_rotation_curve_gradient_task
subroutine node_component_hot_halo_standard_node_merger
subroutine node_component_spheroid_standard_rate_compute
subroutine node_component_spheroid_very_simple_rates
function simpleratemassloss
function chandrasekhar1943acceleration
subroutine simpleget
subroutine cole2000get
subroutine simpleget
subroutine cole2000get

subroutine satellite_orbit_extremum_phase_space_coordinates
function sphericalsymmetrytidaltensorradial
function zentner2005masslossrate
function blitz2006rate
subroutine krumholz2009surfacedensityfactors
subroutine instantaneousrates
function testfuncdouble0
function simpleratemassloss

function node_component_black_hole_noncentral_recoil_escapes
function node_component_black_hole_simple_rotation_curve
subroutine node_component_black_hole_standard_mass_accretion_rate
function node_component_black_hole_standard_potential
function node_component_black_hole_standard_rotation_curve_gradient
subroutine node_component_disk_very_simple_rate_compute
subroutine node_component_hot_halo_cold_mode_node_merger
subroutine node_component_hot_halo_cold_mode_rate_compute
function node_component_hot_halo_cold_mode_enclosed_mass_task
function node_component_hot_halo_cold_mode_rotation_curve_task
subroutine node_component_satellite_orbiting_rate_compute
subroutine node_component_spheroid_very_simple_rate_compute
function intergalacticbackgroundinternalupdate
function simpleratemassloss
subroutine baugh2005get
subroutine verysimpleget
function cole2000halfmassradiusroot
subroutine standardget
function equivalent_circular_orbit_solver
function tracedarkmatterinversecmfradial
function gnedin1999heatingrate

function outflowrateintegrand
function kennicuttschmidtrate
function extendedschmidtrate

subroutine halomodelgenerateperform
function simpleratemassloss
function intergalacticmediumstateevolveupdate

file: galactic_structure.potential.F90
Description: Contains a module which implements calculations of gravitationl potential.
Code lines: 183

module: galactic_structure_potentials
Description: Implements calculations of the gravitational potential.
Code lines: 147
Contained by: file galactic_structure.potential.F90
Modules used: galactic_structure_options, kind_numbers
Used by: subroutine galacticus_calculations_-reset, function node_component_black_hole_-noncentral_recoil_escapes, function node_component_black_hole_-standard_recoil_escapes, function equivalent_circular_orbit_-solver, function extremum_solver, function satellite_orbit_convert_to_-current_potential, subroutine satellite_orbit_extremum_-phase_space_coordinates

function: component_potential
Description: Unary function returning the potential in a component. Suitable for mapping over components.
Code lines: 8
Contained by: module galactic_structure_potentials
Modules used: galacticus_nodes

function: galactic_structure_potential
Description: Solve for the gravitational potential at a given radius. Assumes the galactic structure has already been computed.
Code lines: 100
Contained by: module galactic_structure_potentials
Modules used: dark_matter_halo_scales, dark_matter_profile_structure_tasks, galacticus_nodes, node_component_black_hole_simple_-structure, node_component_black_hole_standard_-structure_tasks

subroutine: galactic_structure_potential_standard_reset
Description: Reset calculations for galactic structure potentials.
Code lines: 9
Contained by: module galactic_structure_potentials
Modules used: galacticus_nodes

file: galactic_structure.radius_definition.F90
Description: Contains a module which provides parsing of radii definitions used in output specifiers.
Code lines: 189

module: galactic_structure_radii_definitions
Description: Provides parsing of radii definitions used in output specifiers.

Code lines: 167
Contained by: file galactic_structure.radius_definition.F90
Modules used: iso_varying_string
Used by: file nodes.property_extractor.density.F90; function densityprofileconstructorinternal; function densityprofileextract; file nodes.property_extractor.projected_density.F90; function projecteddensityconstructorinternal; function projecteddensityextract; file nodes.property_extractor.rotation_curve.F90; function rotationcurveconstructorinternal; function rotationcurveextract; file nodes.property_extractor.velocity_dispersion.F90; function velocitydispersionconstructorinternal; function velocitydispersionextract

subroutine: galactic_structure_radii_definition_decode
Description: Decode a set of radii descriptors and return the corresponding specifiers.
Code lines: 121
Contained by: module galactic_structure_radii_definitions
Modules used: galactic_structure_options; galacticus_error; galacticus_nodes; stellar_luminosities_structure; string_handling

type: radiusspecifier
Description: Type used for specifying radii definitions used in output specifiers.
Code lines: 5
Contained by: module galactic_structure_radii_definitions

file: galactic_structure.radius_solver.F90
Description: Contains a module which implements a class for computing radii of galactic components (or more general components).
Code lines: 62

module: galactic_structure_solvers
Description: Implements a class for calculations of sizes of galactic components (or more general components).
Code lines: 40
Contained by: file galactic_structure.radius_solver.F90
Modules used: galacticus_nodes
Used by: subroutine galacticus_function_classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; file merger_trees.evolver.standard.F90; module merger_trees_evolve_node

file: galactic_structure.radius_solver.equilibrium.F90
Description: Implementation of an "equilibrium" solver for galactic structure.
Code lines: 469
Modules used: dark_matter_profiles; dark_matter_profiles_dmo

subroutine: `equilibriumautohook`
Description: Attach to various event hooks.
Code lines: 11
Contained by: file galactic_structure.radius_solver.equilibrium.F90
Modules used: events_hooks

function: `equilibriumconstructorinternal`
Description: Internal constructor for the `equilibrium` galactic structure solver class.
Code lines: 11
Contained by: file galactic_structure.radius_solver.equilibrium.F90

function: `equilibriumconstructorparameters`
Description: Constructor for the `equilibrium` galactic structure solver class which takes a parameter set as input.
Code lines: 51
Contained by: file galactic_structure.radius_solver.equilibrium.F90
Modules used: input_parameters

subroutine: `equilibriumdestructor`
Description: Destructor for the `equilibrium` galactic structure solver class.
Code lines: 13
Contained by: file galactic_structure.radius_solver.equilibrium.F90
Modules used: events_hooks

subroutine: `equilibriumrevert`
Description: Revert radii for the equilibrium galactic structure solve.
Code lines: 10
Contained by: file galactic_structure.radius_solver.equilibrium.F90

subroutine: `equilibriumsolve`
Description: Solve for the structure of galactic components.
Code lines: 271
Contained by: file galactic_structure.radius_solver.equilibrium.F90
Modules used: galacticus_display, galacticus_error, node_component_basic_standard, node_component_dark_matter_profile_scale, node_component_disk_standard, node_component_disk_very_simple_size, node_component_spheroid_standard, node_component_spheroid_very_simple

subroutine: `radiussolve`
Description: Solve for the equilibrium radius of the given component.
Code lines: 152
Contained by: subroutine equilibriumsolve
Modules used: galactic_structure_options, galactic_structure_rotation_curves, galacticus_display, galacticus_error, iso_varying_string, memory_management, numerical_constants_physical, string_handling

subroutine: `equilibriumsolvehook`
Description: Hookable wrapper around the solver.
Code lines: 14
Contained by: file `galactic_structure.radius_solver.equilibrium.F90`
Modules used: `galacticus_error`

subroutine: `equilibriumsolveprederiativehook`
Description: Hookable wrapper around the solver for pre-derivative events.
Code lines: 16
Contained by: file `galactic_structure.radius_solver.equilibrium.F90`
Modules used: `galacticus_error` `galacticus_nodes`

interface: `galacticstructuresolverequilibrium`
Description: Constructors for the `equilibrium` galactic structure solver class.
Code lines: 4
Contained by: file `galactic_structure.radius_solver.equilibrium.F90`

file: `galactic_structure.radius_solver.fixed.F90`
Description: Implementation of a "fixed" solver for galactic structure (no self-gravity of baryons, and size simply scales in proportion to specific angular momentum).
Code lines: 303
Modules used: `dark_matter_halo_scales` `dark_matter_profiles_dmo`

subroutine: `fixedautohook`
Description: Attach to various event hooks.
Code lines: 11
Contained by: file `galactic_structure.radius_solver.fixed.F90`
Modules used: `events_hooks`

function: `fixedconstructorinternal`
Description: Internal constructor for the `fixed` galactic structure solver class.
Code lines: 17
Contained by: file `galactic_structure.radius_solver.fixed.F90`
Modules used: `galacticus_error` `galacticus_nodes`

function: `fixedconstructorparameters`
Description: Constructor for the `fixed` galactic structure solver class which takes a parameter set as input.
Code lines: 36
Contained by: file `galactic_structure.radius_solver.fixed.F90`
Modules used: `input_parameters`

subroutine: `fixeddestructor`
Description: Destructor for the `fixed` galactic structure solver class.
Code lines: 13
Contained by: file `galactic_structure.radius_solver.fixed.F90`
Modules used: `events_hooks`

subroutine: `fixedrevert`
Description: Revert radii for the fixed galactic structure solve. Not necessary for this algorithm.
Code lines: 8
Contained by: file `galactic_structure.radius_solver.fixed.F90`

subroutine: `fixedsolve`
Description: Solve for the structure of galactic components assuming no self-gravity of baryons, and that size simply scales in proportion to specific angular momentum.
Code lines: 112
Contained by: file `galactic_structure.radius_solver.fixed.F90`
Modules used: `node_component_basic_standard` `node_component_dark_matter_profile_scale` `node_component_disk_standard` `node_component_disk_very_simple_size` `node_component_spheroid_standard` `node_component_spheroid_very_simple`

subroutine: `radiussolve`
Description: Solve for the equilibrium radius of the given component.
Code lines: 28
Contained by: subroutine `fixedsolve`
Modules used: `galacticus_nodes`

subroutine: `fixedsolvehook`
Description: Hookable wrapper around the solver.
Code lines: 14
Contained by: file `galactic_structure.radius_solver.fixed.F90`
Modules used: `galacticus_error`

subroutine: `fixedsolveprederiativehook`
Description: Hookable wrapper around the solver.
Code lines: 16
Contained by: file `galactic_structure.radius_solver.fixed.F90`
Modules used: `galacticus_error`

interface: `galacticstructuresolverfixed`
Description: Constructors for the `fixed` galactic structure solver class.
Code lines: 4
Contained by: file `galactic_structure.radius_solver.fixed.F90`

file: `galactic_structure.radius_solver.linear.F90`
Description: Implementation of a "linear" solver for galactic structure (no self-gravity of baryons, and size simply scales in proportion to specific angular momentum).
Code lines: 247
Modules used: `dark_matter_halo_scales`

interface: `galacticstructuresolverlinear`
Description: Constructors for the `linear` galactic structure solver class.
Code lines: 4
Contained by: file `galactic_structure.radius_solver.linear.F90`

subroutine: `linearautohook`
Description: Attach to various event hooks.
Code lines: 11
Contained by: file `galactic_structure.radius_solver.linear.F90`
Modules used: `events_hooks`

function: `linearconstructorinternal`
Description: Internal constructor for the `linear` galactic structure solver class.
Code lines: 8
Contained by: file `galactic_structure.radius_solver.linear.F90`

function: `linearconstructorparameters`
Description: Constructor for the `linear` galactic structure solver class which takes a parameter set as input.
Code lines: 14
Contained by: file `galactic_structure.radius_solver.linear.F90`
Modules used: `input_parameters`

subroutine: `lineardestructor`
Description: Destructor for the `linear` galactic structure solver class.
Code lines: 12
Contained by: file `galactic_structure.radius_solver.linear.F90`
Modules used: `events_hooks`

subroutine: `linearrevert`
Description: Revert radii for the linear galactic structure solve. Not necessary for this algorithm.
Code lines: 8
Contained by: file `galactic_structure.radius_solver.linear.F90`

subroutine: `linearsolve`
Description: Solve for the structure of galactic components assuming no self-gravity of baryons, and that size simply scales in proportion to specific angular momentum.
Code lines: 102
Contained by: file `galactic_structure.radius_solver.linear.F90`
Modules used: `node_component_basic_standard` `node_component_dark_matter_profile_scale` `node_component_disk_standard` `node_component_disk_very_simple_size` `node_component_spheroid_standard` `node_component_spheroid_very_simple`

subroutine: `radiussolve`
Description: Solve for the equilibrium radius of the given component.
Code lines: 19
Contained by: subroutine `linearsolve`

subroutine: `linearsolvehook`
Description: Hookable wrapper around the solver.
Code lines: 14
Contained by: file `galactic_structure.radius_solver.linear.F90`
Modules used: `galacticus_error`

subroutine: `linearsolveprederiativehook`
Description: Hookable wrapper around the solver.
Code lines: 16
Contained by: file `galactic_structure.radius_solver.linear.F90`
Modules used: `galacticus_error`

file: `galactic_structure.radius_solver.simple.F90`
Description: Implementation of a simple solver for galactic structure (self-gravity of baryons is ignored).
Code lines: 277
Modules used: `dark_matter_profiles_dmo`

interface: `galacticstructuresolversimple`
Description: Constructors for the `simple` galactic structure solver class.
Code lines: 4
Contained by: file `galactic_structure.radius_solver.simple.F90`

subroutine: `simpleautohook`
Description: Attach to various event hooks.
Code lines: 11
Contained by: file `galactic_structure.radius_solver.simple.F90`
Modules used: `events_hooks`

function: `simpleconstructorinternal`
Description: Internal constructor for the `simple` galactic structure solver class.
Code lines: 9
Contained by: file `galactic_structure.radius_solver.simple.F90`

function: `simpleconstructorparameters`
Description: Constructor for the `simple` galactic structure solver class which takes a parameter set as input.
Code lines: 23
Contained by: file `galactic_structure.radius_solver.simple.F90`
Modules used: `input_parameters`

subroutine: `simpledestructor`
Description: Destructor for the `simple` galactic structure solver class.
Code lines: 12
Contained by: file `galactic_structure.radius_solver.simple.F90`
Modules used: `events_hooks`

subroutine: `simplerevert`
Description: Revert radii for the simple galactic structure solve. Not necessary for this algorithm.
Code lines: 8
Contained by: file `galactic_structure.radius_solver.simple.F90`

subroutine: `simplesolve`
Description: Solve for the structure of galactic components.
Code lines: 123

Contained by: file galactic_structure.radius_solver.simple.F90
Modules used: galacticus_error, node_component_basic_standard, node_component_dark_matter_profile_scale, node_component_disk_standard, node_component_disk_very_simple_size, node_component_spheroid_standard, node_component_spheroid_very_simple

subroutine: radiussolve
Description: Solve for the equilibrium radius of the given component.
Code lines: 20
Contained by: subroutine simplesolve

subroutine: simplesolvehook
Description: Hookable wrapper around the solver.
Code lines: 14
Contained by: file galactic_structure.radius_solver.simple.F90
Modules used: galacticus_error

subroutine: simplesolveprederiativehook
Description: Hookable wrapper around the solver.
Code lines: 16
Contained by: file galactic_structure.radius_solver.simple.F90
Modules used: galacticus_error

file: galactic_structure.rotation_curve.F90
Description: Contains a module which implements calculations of the rotation curve as a specified radius.
Code lines: 117

module: galactic_structure_rotation_curves
Description: Implements calculations of the rotation curve as a specified radius.
Code lines: 95
Contained by: file galactic_structure.rotation_curve.F90
Modules used: galactic_structure_options
Used by: function standardgrowthrate, subroutine radiussolve, function galactic_structure_rotation_curve_gradient, function rotationcurveextract, function velocitydispersionlambdarintegrand1, function velocitydispersionlambdarintegrand2, function gnedin1999heatingrate, function simpleratemassloss, function simpleratemassloss

function: component_rotation_curve
Description: Unary function returning the squared rotation curve in a component. Suitable for mapping over components.
Code lines: 8
Contained by: module galactic_structure_rotation_curves
Modules used: galacticus_nodes

function: galactic_structure_rotation_curve

Description: Solve for the rotation curve a given radius. Assumes that galactic structure has already been solved for.
Code lines: 70
Contained by: module galactic_structure_rotation_curves
Modules used: dark_matter_profile_structure_tasks, galacticus_nodes, hot_halo_mass_distributions, node_component_black_hole_simple_-structure, node_component_black_hole_standard_-structure_tasks, node_component_hot_halo_cold_mode_-structure_tasks

file: galactic_structure.rotation_curve.gradient.F90
Description: Contains a module which implements calculations of the gradient of the rotation curve.
Code lines: 134

module: galactic_structure_rotation_curve_gradients
Description: Implements calculations of the rotation curve gradient
Code lines: 98
Contained by: file galactic_structure.rotation_curve.gradient.F90
Modules used: galactic_structure_options
Used by: function standardgrowthrate

function: component_rotation_curve_gradient
Description: Unary function returning the gradient of the squared rotation curve in a component. Suitable for mapping over components.
Code lines: 8
Contained by: module galactic_structure_rotation_curve_gradients
Modules used: galacticus_nodes

function: galactic_structure_rotation_curve_gradient
Description: Solve for the rotation curve gradient at a given radius. Assumes the galactic structure has already been computed.
Code lines: 73
Contained by: module galactic_structure_rotation_curve_gradients
Modules used: dark_matter_profile_structure_tasks, galactic_structure_rotation_curves, galacticus_nodes, hot_halo_mass_distributions, node_component_black_hole_simple_-structure, node_component_black_hole_standard_-structure_tasks, node_component_hot_halo_cold_mode_-structure_tasks

file: galactic_structure.surface_density.F90
Description: Contains a module which implements calculations of the surface density at a specific position.
Code lines: 103

module: galactic_structure_surface_densities
Description: Implements calculations of the surface density at a specific position.
Code lines: 81
Contained by: file galactic_structure.surface_density.F90
Modules used: galactic_structure_options

Used by: function velocitydispersionlambdarintegrand1; function velocitydispersionlambdarintegrand2; function simpleratemassloss; function outflowrateintegrand; function blitz2006rate; function kennicuttschmidtrate; subroutine krumholz2009surfacedensityfactors; function extendedschmidtrate

function: `component_surface_density`

Description: Unary function returning the surface density in a component. Suitable for mapping over components.
Code lines: 8
Contained by: module galactic_structure_surface_densities
Modules used: galacticus_nodes

function: `galactic_structure_surface_density`

Description: Compute the density (of given `massType`) at the specified `position`. Assumes that galactic structure has already been computed.
Code lines: 55
Contained by: module galactic_structure_surface_densities
Modules used: coordinate_systems; galacticus_error; galacticus_nodes

file: `galactic_structure.velocity_dispersions.F90`

Description: Contains a module which implements calculations of the velocity dispersions in isotropic spherical systems by solving the Jeans equation.
Code lines: 92

module: `galactic_structure_velocity_dispersions`

Description: Implements calculations of the velocity dispersions in isotropic spherical systems by solving the Jeans equation.
Code lines: 67
Contained by: file galactic_structure.velocity_dispersions.F90
Modules used: galacticus_nodes
Used by: function standardgrowthrate; function velocitydispersionextract; function velocitydispersionvelocitysurfacedensityintegrand

function: `galactic_structure_velocity_dispersion`

Description: Returns the velocity dispersion of the specified `componentType` in `thisNode` at the given `radius`.
Code lines: 36
Contained by: module galactic_structure_velocity_dispersions
Modules used: fgsl; galactic_structure_densities; galactic_structure_enclosed_masses; galactic_structure_options; numerical_integration

function: `velocity_dispersion_integrand`

Description: Integrand function used for finding velocity dispersions using Jeans equation.
Code lines: 14

Contained by: module galactic_structure_velocity_dispersions
Modules used: galactic_structure_densities galactic_structure_enclosed_masses numerical_constants_physical

file: galacticus.banner.F90
Description: Contains a module which displays a banner for Galacticus.
Code lines: 61

module: galacticus_banner
Description: Displays a banner for Galacticus.
Code lines: 39
Contained by: file galacticus.banner.F90
Used by: program galacticus

subroutine: galacticus_banner_show
Description: Displays the Galacticus banner.
Code lines: 29
Contained by: module galacticus_banner
Modules used: galacticus_display mpi_utilities

file: galacticus.calculations_reset.F90
Description: Contains a module which handles resetting of calculations before a new or updated node is processed.
Code lines: 81

module: galacticus_calculations_resets
Description: Handles resetting of calculations before a new or updated node is processed.
Code lines: 59
Contained by: file galacticus.calculations_reset.F90
Used by: function nbodyerrorsdistributionaveraged
function ludlow2016radius
function concentrationradius
function powerspectrumonehalointegrand
subroutine fixedmassconstruct
subroutine standardevolve
subroutine analyzeroutput
function haloradiusrootfunction
subroutine halomassfunctionperform
subroutine nbodyerrorstabulate
function concentrationmassroot
subroutine functions_global_set
function powerspectrumtwohalointegrand
subroutine standardderivativescompute
subroutine particulateoperate
subroutine standardoutput
function virial_density_contrast_percolation_solver
subroutine halomodelgenerateperform

subroutine: galacticus_calculations_reset
Description: Calls any routines required to reset all calculation for a new or updated node.
Code lines: 43
Contained by: module galacticus_calculations_resets
Modules used: galactic_structure_potentials galacticus_nodes node_component_disk_standard node_component_hot_halo_standard node_component_hot_halo_very_simple

file: `galacticus.display.F90`

Description: Contains a module which implements outputting of formatted, indented messages at various vebosity levels from Galacticus.

Code lines: 384

module: `galacticus_display`

Description: Implements outputting of formatted, indented messages at various vebosity levels from Galacticus.

Code lines: 362

Contained by: file galacticus.display.F90

Modules used: iso_varying_string

Used by:

subroutine hopkins2007buildfile	program benchmark_stellar_populations_luminosities
function standardgrowthrate	subroutine ciefilereadfile
subroutine atomiccieclOudytabulate	subroutine ciefilereadfile
subroutine atomiccieclOudytabulate	function ludlow2016radius
subroutine einastofourierprofiletablemake	subroutine einastofreefalltabulate
subroutine einastoradialvelocitydispersiontabulate	function heateddensity
function node_branch_jump	subroutine inter_tree_event_post_evolve
function node_pull_from_tree	function node_push_from_tree
function node_subhalo_promotion	function galactic_structure_radius_enclosing_density
function galactic_structure_radius_enclosing_mass	subroutine equilibriumsolve
subroutine radiussolve	subroutine galacticus_banner_show
subroutine galacticus_verbosity_set_from_parameters	subroutine galacticus_warn_char
subroutine windowread	subroutine caputi2011ukidssudsrandomsinitialize
subroutine liwhite2009sdssrandomsinitialize	subroutine martin2010alfalfarandomsinitialize
subroutine fullskywindowfunctions	subroutine randompointswindowfunctions
function powerspectrumonehalotimeintegrand	function powerspectrumtwohalotimeintegrand
function font2008radiusstripped	subroutine interface_camb_initialize
subroutine interface_cloudy_initialize	subroutine interface_cloudy_cie_tabulate
subroutine interface_fsps_initialize	subroutine interface_recfast_initialize
function recfastconstructorinternal	function betaprofileconstructorinternal
function parkinsoncolehellyprobabilitybound	function generalizedpressschechterfractionsubresolution
function generalizedpressschechtermassbranch	sampleddistributionconstructorparameters

function fullyspecifiedconstruct
subroutine readassignsplitforestevents
function readconstructorinternal
subroutine readscanforbranchjumps
subroutine readtimingreport
subroutine sussingasciiopen
subroutine sussinghdf5load
subroutine sussinghdf5open
subroutine galacticusopen
subroutine satellitemergerprocess
function standardtimeevolveto
subroutine standardevolve
subroutine standardpromote
subroutine augmentoperate
function galaxypopulationconstructorparameters
function gaussianregressionevaluate
function halomassfunctionconstructorinternal
function massfunctionconstructorinternal
subroutine paircountsoperate
subroutine latentintegrator
subroutine points_survey_geometry
subroutine abundances_dump
subroutine history_dump
subroutine kepler_orbits_dump
subroutine nodecomponentagestatisticscreate
subroutine nodecomponentagestatisticsnullserializeascii
subroutine nodecomponentagestatisticsstandardserializeascii
subroutine nodecomponentbasicdumpascii
subroutine nodecomponentbasicnonevolvingserializeascii
subroutine nodecomponentbasicserializeascii
subroutine nodecomponentbasicstandardserializeascii
subroutine readassignscaleradii
subroutine readbuilddescendentpointers
subroutine readrootnodeaffinitiesinitial
subroutine readtimeuntilmergingsubresolution
subroutine sussingasciiload
subroutine sussingtreeindicesread
function decodeunits
subroutine galacticusimport
subroutine evolve_to_time_report
subroutine standardevolve
subroutine standarderrorhandler
subroutine standardmerge
function augmentaccepttree
subroutine particulateoperate
function galaxypopulationevaluate
function gaussianregressionwillevaluate
function independentlikelihoodssequantialevaluate
subroutine environmentaloverdensityoperate
subroutine selfboundoperate
function multivectorizedcompositetrapezoidal1devaluate
function root_finder_find
subroutine chemicals_dump
subroutine history_long_integer_dump
subroutine merger_tree_data_structure_-read_ascii
subroutine nodecomponentagestatisticsdumpascii
subroutine nodecomponentagestatisticsserializeascii
subroutine nodecomponentbasiccreate
subroutine nodecomponentbasicextendedtrackingserializeascii
subroutine nodecomponentbasicnullserializeascii
subroutine nodecomponentbasicstandardextendedserializeascii
subroutine nodecomponentbasicstandardtrackingserializeascii

subroutine nodecomponentblackholecreate

subroutine nodecomponentblackholedumpascii

subroutine nodecomponentblackholenoncentralserializeascii

subroutine nodecomponentblackholenullserializeascii

subroutine nodecomponentblackholeserializeascii

subroutine nodecomponentblackholesimpleserializeascii

subroutine nodecomponentblackholestandardserializeascii

subroutine nodecomponentdarkmatterprofilecreate

subroutine nodecomponentdarkmatterprofiledumpascii

subroutine nodecomponentdarkmatterprofilenullserializeascii

subroutine nodecomponentdarkmatterprofilescalepresetserializeascii

subroutine nodecomponentdarkmatterprofilescaleserializeascii

subroutine nodecomponentdarkmatterprofilescaleshapeserializeascii

subroutine nodecomponentdarkmatterprofileserializeascii

subroutine nodecomponentdiskcreate

subroutine nodecomponentdiskdumpascii

subroutine nodecomponentdisknullserializeascii

subroutine nodecomponentdiskserializeascii

subroutine nodecomponentdiskstandardserializeascii

subroutine nodecomponentdiskverysimpleserializeascii

subroutine nodecomponentdiskverysimplesizeserializeascii

subroutine nodecomponentdynamicsstatisticsbarsserializeascii

subroutine nodecomponentdynamicsstatisticscreate

subroutine nodecomponentdynamicsstatisticsdumpascii

subroutine nodecomponentdynamicsstatisticsnullserializeascii

subroutine nodecomponentdynamicsstatisticsserializeascii

subroutine nodecomponentformationtimecole2000serializeascii

subroutine nodecomponentformationtimecreate

subroutine nodecomponentformationtimedumpascii

subroutine nodecomponentformationtimemassfractionserializeascii

subroutine nodecomponentformationtimenullserializeascii

subroutine nodecomponentformationtimeserializeascii

subroutine nodecomponenthosthistorycreate

subroutine nodecomponenthosthistorydumpascii

subroutine nodecomponenthosthistorynullserializeascii

subroutine nodecomponenthosthistoryserializeascii

subroutine nodecomponenthosthistorystandardserializeascii

subroutine nodecomponenthothalocoldmodeserializeascii

subroutine nodecomponenthothalocreate

subroutine nodecomponenthothalodumpascii

subroutine nodecomponenthothalonullserializeascii

subroutine nodecomponenthothaloutflowtrackingserializeascii

subroutine nodecomponenthothaloserializeascii

subroutine nodecomponenthothalostandardserializeascii

subroutine nodecomponenthothaloverysimpledelayedserializeascii

subroutine nodecomponenthothaloverysimpleserializeascii

subroutine nodecomponentindicescreate

subroutine nodecomponentindicesdumpascii

subroutine nodecomponentindicesnullserializeascii

subroutine nodecomponentindicesserializeascii

subroutine nodecomponentindicesstandardserializeascii

subroutine nodecomponentinteroutputcreate

subroutine nodecomponentinteroutputdumpascii

subroutine nodecomponentinteroutputnullserializeascii

subroutine nodecomponentinteroutputserializeascii

subroutine nodecomponentinteroutputstandardserializeascii

subroutine nodecomponentmassflowstatisticscreate

subroutine nodecomponentmassflowstatisticsdumpascii

subroutine nodecomponentmassflowstatisticsnullserializeascii

subroutine nodecomponentmassflowstatisticsserializeascii

subroutine nodecomponentmassflowstatisticsstandardserializeascii

subroutine nodecomponentmergingstatisticscreate

subroutine nodecomponentmergingstatisticsdumpascii

subroutine nodecomponentmergingstatisticsmajorserializeascii

subroutine nodecomponentmergingstatisticsnullserializeascii

subroutine nodecomponentmergingstatisticsrecentserializeascii

subroutine nodecomponentmergingstatisticsserializeascii

subroutine nodecomponentmergingstatisticsstandardserializeascii

subroutine nodecomponentnbodycreate

subroutine nodecomponentnbodydumpascii

subroutine nodecomponentnbodygenericserializeascii

subroutine nodecomponentnbodynullserializeascii

subroutine nodecomponentnbodyserializeascii

subroutine nodecomponentpositioncreate

subroutine nodecomponentpositiondumpascii

subroutine nodecomponentpositionnullserializeascii

subroutine nodecomponentpositionpresetorphansserializeascii

subroutine nodecomponentpositionpresetserializeascii

subroutine nodecomponentpositionserializeascii

subroutine nodecomponentpositiontracedarkmatterserializeascii

subroutine nodecomponentsatellitecreate

subroutine nodecomponentsatellitedumpascii

subroutine nodecomponentsatellitenullserializeascii

subroutine nodecomponentsatelliteorbitingserializeascii

subroutine nodecomponentsatellitepresetserializeascii

subroutine nodecomponentsatelliteserializeascii

subroutine nodecomponentsatellitestandardserializeascii

subroutine nodecomponentsatelliteverysimpleserializeascii

subroutine nodecomponentspheroidcreate

subroutine nodecomponentspheroiddumpascii

subroutine nodecomponentspheroidnullserializeascii

subroutine nodecomponentspheroidserializeascii

subroutine nodecomponentspheroidstandardserializeascii
subroutine nodecomponentspheroidverysimpleserializeascii
subroutine nodecomponentspincreate
subroutine nodecomponentspindumpascii
subroutine nodecomponentspinnullserializeascii
subroutine nodecomponentspinpreset3dserializeascii
subroutine nodecomponentspinpresetserializeascii
subroutine nodecomponentspinrandomserializeascii
subroutine nodecomponentspinserializeascii
subroutine nodecomponentspinvitvitskaserializeascii
subroutine tree_node_remove_from_host
subroutine tree_node_remove_from_mergee
subroutine treenodeserializeascii
subroutine treenodeserializexml
subroutine node_component_disk_standard_post_step
subroutine node_component_disk_standard_state_retrieve
subroutine node_component_disk_standard_state_store
subroutine node_component_disk_very_simple_post_evolve
subroutine node_component_disk_very_simple_post_step
subroutine node_component_satellite_preset_inter_tree_postprocess
subroutine node_component_satellite_preset_orphanize
subroutine node_component_satellite_preset_satellite_host_change
subroutine node_component_spheroid_standard_post_step
subroutine node_component_spheroid_standard_state_retrieve
subroutine node_component_spheroid_standard_state_store
subroutine node_component_spheroid_very_simple_post_step
subroutine stellar_luminosities_dump
subroutine stellar_luminosities_state_restore
subroutine stellar_luminosities_state_store
subroutine tensor_r2_d3_sym_dump
subroutine tensor_r2_d3_sym_dump_raw
subroutine tensor_r2_d3_sym_read_raw
function gelmanrubinisconverged
function adaptivegamma
function adaptiveexponent
subroutine annealeddifferentialevolutionupdate
function differentialevolutionconstructorparameters
subroutine differentialevolutionposterior
subroutine differentialevolutionsimulate
function particleswarmconstructorparameters
subroutine particleswarmposterior
subroutine particleswarmsimulate
subroutine tempereddifferentialevolutionupdate
subroutine correlationcorrelationlengthcompute
subroutine resumeinitialize
function intergalacticbackgroundinternalupdate
subroutine cole2000get
subroutine covington2008get
function jiang2008constructorparameters
function fixedorbit
subroutine satellite_move_to_new_host
subroutine statistics_points_correlation

subroutine statistics_points_power_spectrum
function standardinterpolate
subroutine farahifileread
subroutine farahiratetabulate
subroutine farahimidpointratetabulate
function zhanghuihighorderprobability
function powerlawconstructorparameters
subroutine filereadfile
subroutine localgroupdatabaseperform
subroutine buildtoolcambperform
subroutine buildtoolfspsperform
subroutine catalogprojectedcorrelationfunctionperform
subroutine evolveforestsperform
subroutine halomassfunctionperform
subroutine halomodelgenerateperform
subroutine intergalacticmediumstateperform
subroutine massfunctioncovariancelsswindowfunction
subroutine mergertreefilebuilderperform
subroutine nbodyanalyzeperform
subroutine powerspectraperform
program test_diemerkravtsov2014_concentration
program tests_io_xml
program test_nfw96_concentration_dark_energy
program test_ode_solver
program test_zhao2009_flat
program test_zhao2009_open
program test_accretion_disks
program test_black_hole_fundamentals
program tests_comoving_distance
program test_correa2015_concentration
program tests_cosmic_age
program test_dark_matter_profiles
program test_dark_matter_profiles_heated
subroutine stellar_population_luminosity_tabulate
function farahiconstructorinternal
function farahiprobability
function farahimidpointprobability
function zhanghuiprobability
function cosmicemuvalue
subroutine make_table
subroutine percolationtabulate
subroutine agnspectrahopkins2008buildfileperform
subroutine buildtoolcloudyperform
subroutine buildtoolrecfastperform
subroutine conditionalmassfunctionperform
subroutine excursionsetsperform
subroutine halomodelprojectedcorrelationfunctionperform
subroutine halospindistributionperform
subroutine massfunctioncovariancelssangularspectrum
subroutine massfunctioncovarianceperform
subroutine multiperform
subroutine posteriorsampleperform
subroutine reportperform
program tests_io_hdf5
program test_mpi
program test_ode_solver
program test_prada2011_concentration
program test_zhao2009_dark_energy
program test_abundances
program test_array_monotonicity
program tests_bug745815
program test_comparison
program test_cooling_functions
program test_dark_matter_halo_radius_enclosing_mass
program test_dark_matter_profiles_generic
program test_differentiation

program test_gaunt_factors
program tests_halo_mass_function_-tinker
program test_hashes_cryptographic
program test_initial_mass_functions
module test_integration_functions
program test_interpolation_2d
program tests_kepler_orbits
program tests_linear_growth_-cosmological_constant
program tests_linear_growth_open
program test_make_ranges
program test_mass_distributions
program test_math_distributions
program test_meshes
program test_nodes
function testfuncdouble0
program test_parameters
program test_random
program test_root_finding
program test_search
program test_sort
program test_inoue2014

program tests_spherical_collapse_-dark_energy_omega_zero_point_six
program tests_spherical_collapse_-dark_energy_omega_half
program tests_spherical_collapse_-dark_energy_lambda
program tests_spherical_collapse_flat

program tests_spherical_collapse_open
program test_stellar_populations_-luminosities
program test_tables
program tests_transfer_functions
program test_vectors

subroutine io_hdf5_close
subroutine inputparameterscheckparameters
subroutine allocatearray_character_1d

program test_coordinate_systems
program test_hashes

program test_perfect_hashes
program test_integration
program test_integration2
program test_interpolation
program tests_linear_growth_eds
program tests_linear_growth_dark_-energy
program test_locks
program test_correa2015_mah
program test_math_fast
program test_math_special_functions
program test_multi_counters
subroutine test_node_task
subroutine testvoidfunc
program tests_power_spectrum
program tests_regular_expressions
module test_root_finding_functions
program tests_sigma
program test_sort_topological
program tests_spherical_collapse_-dark_energy_eds
program tests_spherical_collapse_-dark_energy_omega_zero_point_eight
program tests_spherical_collapse_-dark_energy_omega_two_thirds
program tests_spherical_collapse_-dark_energy_open
program tests_spherical_collapse_-nonlinear
program test_stellar_populations
program test_string_utilities

program test_tensors
program tests_tree_branch_destroy
function intergalacticmediumstateevolveupdate
subroutine io_hdf5_flush
function inputparametersconstructornode

subroutine allocatearray_character_1d_-kind_int8

subroutine allocatearray_complex_c_-double_complex_3d
subroutine allocatearray_complex_c_-double_complex_3d_kind_int8
subroutine allocatearray_double_-precision_1d
subroutine allocatearray_double_-precision_1d_kind_int8
subroutine allocatearray_double_-precision_2d
subroutine allocatearray_double_-precision_2d_kind_int8
subroutine allocatearray_double_-precision_3d
subroutine allocatearray_double_-precision_3d_kind_int8
subroutine allocatearray_double_-precision_4d
subroutine allocatearray_double_-precision_4d_kind_int8
subroutine allocatearray_double_-precision_5d
subroutine allocatearray_double_-precision_5d_kind_int8
subroutine allocatearray_double_-precision_6d
subroutine allocatearray_double_-precision_6d_kind_int8
subroutine allocatearray_integer_1d
subroutine allocatearray_integer_1d_-kind_int8
subroutine allocatearray_integer_2d
subroutine allocatearray_integer_2d_-kind_int8
subroutine allocatearray_integer_kind_-int8_1d
subroutine allocatearray_integer_kind_-int8_1d_kind_int8
subroutine allocatearray_integer_kind_-int8_2d
subroutine allocatearray_integer_kind_-int8_2d_kind_int8
subroutine allocatearray_logical_1d
subroutine allocatearray_logical_1d_-kind_int8
subroutine allocatearray_logical_2d
subroutine allocatearray_logical_2d_-kind_int8
subroutine allocatearray_real_1d
subroutine allocatearray_real_1d_kind_-int8
subroutine allocatearray_real_2d
subroutine allocatearray_real_2d_kind_-int8
subroutine allocatearray_real_kind_-quad_1d
subroutine allocatearray_real_kind_-quad_1d_kind_int8
subroutine deallocatearray_character_1d
subroutine deallocatearray_complex_c_-double_complex_3d
subroutine deallocatearray_double_-precision_1d
subroutine deallocatearray_double_-precision_2d
subroutine deallocatearray_double_-precision_3d
subroutine deallocatearray_double_-precision_4d
subroutine deallocatearray_double_-precision_5d
subroutine deallocatearray_double_-precision_6d
subroutine deallocatearray_integer_1d
subroutine deallocatearray_integer_2d
subroutine deallocatearray_integer_-kind_int8_1d
subroutine deallocatearray_integer_-kind_int8_2d
subroutine deallocatearray_logical_1d
subroutine deallocatearray_logical_2d
subroutine deallocatearray_real_1d
subroutine deallocatearray_real_2d

subroutine deallocatearray_real_kind_-quad_1d
subroutine memory_usage_record
subroutine memory_usage_report
subroutine unit_tests_finish

subroutine: `create_indentation_format`
Description: Create a format for indentation.
Code lines: 25
Contained by: module galacticus_display

subroutine: `galacticus_display_counter`
Description: Displays a percentage counter and bar to show progress.
Code lines: 11
Contained by: module galacticus_display

subroutine: `galacticus_display_counter_clear`
Description: Clears a percentage counter.
Code lines: 11
Contained by: module galacticus_display

subroutine: `galacticus_display_counter_clear_lockless`
Description: Clears a percentage counter.
Code lines: 18
Contained by: module galacticus_display

subroutine: `galacticus_display_counter_lockless`
Description: Displays a percentage counter and bar to show progress.
Code lines: 28
Contained by: module galacticus_display

interface: `galacticus_display_indent`
Code lines: 3
Contained by: module galacticus_display

subroutine: `galacticus_display_indent_char`
Description: Increase the indentation level and display a message.
Code lines: 34
Contained by: module galacticus_display

subroutine: `galacticus_display_indent_varstr`
Description: Increase the indentation level and display a message.
Code lines: 8
Contained by: module galacticus_display

interface: `galacticus_display_message`
Code lines: 3
Contained by: module galacticus_display

subroutine: `galacticus_display_message_char`
Description: Display a message (input as a `character` variable).

Code lines: 29
Contained by: module galacticus_display

subroutine: galacticus_display_message_varstr
Description: Display a message (input as a varying_string variable).
Code lines: 29
Contained by: module galacticus_display

interface: galacticus_display_unindent
Code lines: 3
Contained by: module galacticus_display

subroutine: galacticus_display_unindent_char
Description: Decrease the indentation level and display a message.
Code lines: 34
Contained by: module galacticus_display

subroutine: galacticus_display_unindent_varstr
Description: Decrease the indentation level and display a message.
Code lines: 8
Contained by: module galacticus_display

function: galacticus_verbosity_level
Description: Returns the verbositly level in Galacticus.
Code lines: 6
Contained by: module galacticus_display

subroutine: galacticus_verbosity_level_set
Description: Set the verbosity level.
Code lines: 7
Contained by: module galacticus_display

subroutine: initialize_display
Description: Initialize the module by determining the requested verbosity level.
Code lines: 46
Contained by: module galacticus_display
Modules used: mpi

file: galacticus.display.verbosity.F90
Description: Contains a module which handles setting of verbosity.
Code lines: 50

module: galacticus_display_verbosity
Description: Handle setting of verbosity.
Code lines: 28
Contained by: file galacticus.display.verbosity.F90
Used by: program galacticus

subroutine: galacticus_verbosity_set_from_parameters

Description: Read the parameter that controls the verbosity level, and set that level.
Code lines: 18
Contained by: module galacticus_display_verbosity
Modules used: galacticus_display, input_parameters

file: galacticus.error.F90

Description: Contains a module which implements error reporting for the Galacticus package.
Code lines: 540

module: galacticus_error

Description: Implements error reporting for the Galacticus package.
Code lines: 518
Contained by: file galacticus.error.F90
Modules used: fgsl, hdf5, iso_c_binding, iso_varying_string, semaphores
Used by: program galacticus; function fgabnd; subroutine xermsg; function bertschingerconstructorinternal; function naozbarkana2007constructorinternal; function coldmodecoldmodefraction; function simpleconstructorinternal; function simpleconstructorparameters; function adafadiabaticindexdefault; function adafconstructorinternal; function adafefficiencyradiative; function eddingtonlimitedconstructorparameters; subroutine hopkins2007buildfile; function fileconstructorinternal; subroutine fileloadfile; function switchedconstructorparameters; function abundance_pattern_lookup; function atom_lookup; subroutine atomic_data_initialize; function sutherland1998total; function vanhoof2014total; function scholzwalters1991coolingrate; function arnaud1985rate; function verner1996rate; function standardgrowthrate; function black_hole_frame_dragging_frequency_node; function black_hole_horizon_radius_node; function black_hole_isco_radius_spin; function black_hole_metric_a_factor_node; function black_hole_metric_d_factor_node; function black_hole_static_radius_node; function hydrogennetworkconstructorinternal; subroutine hydrogennetworkrates; subroutine ciefilereadfile; function dynamicaltimeconstructorinternal; subroutine ciefilereadfile; function cmbcomptontemperature; function summationcoolingfunctiondensitylogslope

function summationcoolingfunctiontemperaturelogslope
function betaprofileconstructorinternal
function simpleconstructorinternal
function cutoffconstructorinternal
function velocitymaximumscalingconstructorinternal
function coolingfreefallconstructorinternal
function constantrotationangularmomentumspecific
function formationtimeconstructorparameters
function matterdarkenergydistancecomoving
function matterdarkenergyequationofstatedarkenergy
function matterdarkenergyexponentdarkenergyderivative
function matterdarkenergyhubbleparameterrateofchange
function matterdarkenergytimeatdistancecomoving
function matterlambdadistanceangular
function matterlambdadistancecomovingconvert
subroutine matterlambdaepochvalidate
function matterlambdahubbleparameterepochal
function matterlambdamatterdensityepochal
function matterlambdaomegamatterepochal
function matterlambdatemperaturecmbepochal
function staticuniversecosmictime
function staticuniversedistanceangular
function staticuniversedistancecomovingconvert
function staticuniversedominationepochmatter
subroutine summationdeepcopy
function isothermalconstructorinternal
function whitefrenk1991constructorinternal
function simplescalingconstructorinternal
function haloformationconstructorparameters
function coolingradiusconstructorinternal
function whitefrenk1991constructorinternal
function matterdarkenergycosmictime
function matterdarkenergydistancecomovingconvert
function matterdarkenergyexponentdarkenergy
function matterdarkenergyhubbleparameterepochal
function matterdarkenergyomegadarkenergyepochal
function matterlambdacosmictime
function matterlambdadistancecomoving
function matterlambdadistanceluminosity
function matterlambdaexpansionfactor
function matterlambdahubbleparameterrateofchange
function matterlambdaomegadarkenergyepochal
function matterlambdaomegamatterrateofchange
function matterlambdatimeatdistancecomoving
subroutine staticuniversedensityscalingearlytime
function staticuniversedistancecomoving
function staticuniversedistanceluminosity
subroutine staticuniverseepochvalidate

function staticuniverseequalityepochmattercurvatureu
function staticuniverseequalityepochmatterdarkenergy
function staticuniverseequalityepochmatterradiation
function staticuniverseomegadarkenergyepochal
function staticuniverseomegamatterepochal
function staticuniverseomegamatterrateofchange
function staticuniversetemperaturecmbepochal
function staticuniversetimeatdistancecomoving
function simpleconstructorparameters
function simplehubbleconstant
function simpleomegabaryon
function simpleomegadarkenergy
function simpleomegamatter
function wechsler2002constructorinternal
function zhao2009time
subroutine virialdensitycontrastdefinitiondeepcopy
function virialdensitycontrastdefinitioninternal
subroutine assertpropertiesgettable
function bett2007sample
function nbodyerrorssample
subroutine nbodyerrorstabulate
function massspinintegral
function deltafunctiondistribution
function adiabaticgnedin2004constructorinternal
function darkmatteronlykspace
subroutine summationdeepcopy
function diemerkravtsov2014constructorinternal
function duttonmaccio2014constructorinternaldefined
function duttonmaccio2014constructorinternaltype
function klypin2015concentration
function klypin2015constructorparameters
function ludlow2016fitconcentration
function schneider2015constructorparameters
function ludlow2014constructorinternal
function ludlow2016constructorinternal
function ludlow2016constructorparameters
function ludlow2016radius
function binaryconstructorinternal
function binaryconstructorparameters
function klypin2015constructorinternal
function klypin2015shape
function dark_matter_profile_-potential_task
function burkertconstructorinternal
function einastoconstructorinternal
subroutine einastofourierprofiletablemake
function nfwconstructorinternal
function heatedconstructorinternal
function isothermalpotential
function truncatedconstructorinternal
function truncatedexponentialconstructorinternal
function truncatedexponentialconstructorparameters
function truncatedexponentialdensitylogslope
function eventhookcount
function eventhookfirst
subroutine eventhookunspecifiedattach
subroutine eventhookunspecifieddetach

subroutine inter_tree_event_post_evolve
function node_pull_from_tree
function node_push_from_tree
subroutine galacticus_calculations_-reset_null
subroutine galacticus_state_retrieve_-null
subroutine galacticus_state_store_null
subroutine percolationobjectsdeepcopy_-null
subroutine tasks_evolve_forest_-construct_null
subroutine tasks_evolve_forest_-destruct_null
subroutine tasks_evolve_forest_-perform_null
function virial_density_contrast_-percolation_objects_constructor_null
function virial_density_contrast_-percolation_solver_null
subroutine alldeepcopy
subroutine anydeepcopy
function stellarabsolutemagnitudesconstructorinternal
function stellarabsolutemagnitudesconstructorparameters
function stellarapparentmagnitudesconstructorinternal
function stellarapparentmagnitudesconstructorparameters
function galactic_structure_density
subroutine galactic_structure_-enclosed_mass_defaults
function galactic_structure_radius_-enclosing_density
function galactic_structure_radius_-enclosing_mass
subroutine galactic_structure_radii_-definition_decode
subroutine equilibriumsolve
subroutine radiussolve
subroutine equilibriumsolvehook
subroutine equilibriumsolveprederiativehook
function fixedconstructorinternal
subroutine fixedsolvehook
subroutine fixedsolveprederiativehook
subroutine linearsolvehook
subroutine linearsolveprederiativehook
subroutine simplesolve
subroutine simplesolvehook
subroutine simplesolveprederiativehook
function galactic_structure_surface_-density
subroutine galacticus_error_wait_set_-from_parameters
subroutine meta_tree_timing_initialize
function hivshalomassrelationpadmanabhan2017constructorinternal
subroutine localgroupmassfunctionreduce
function blackholebulgerelationconstructorinternal
function colordistributionsdssconstructorinternal
function concentrationdistributioncdmcococonstructorinternal
function concentrationvshalomasscdmludlow2016constructorinternal
function correlationfunctionconstructorinternal
function correlationfunctionconstructorparameters
function correlationfunctionloglikelihood
subroutine correlationfunctionreduce
subroutine sequencedeepcopy
function disksizeinclinationoperatedistribution

function magnificationcdfintegrand
function identityoperatedistribution
function randomerrornbdycncconstructorinternal
function sequenceoperatedistribution
function spinnbodyerrorsoperatedistribution
function galaxysizessdssconstructorinternal
function luminosityfunctionhalphaconstructorinternal
function luminosityfunctionsobral2013hizelsconstructorinternal
function massfunctionhiconstructorinternal
function massfunctionstellarconstructorparameters
function massfunctionstellarukidssudsconstructorinternal
function massfunctionstellarviperssconstructorinternal
function massmetallicityandrews2013constructorinternal
function meanfunction1dconstructorparameters
subroutine meanfunction1dreduce
subroutine multireduce
function csmlgyangulardistanceconstructorinternal
function csmlgyluminositydistanceconstructorinternal
subroutine sequencedeepcopy
function squarerootoperate
function scatterfunction1dloglikelihood
function spindistributionbett2007constructorinternal
subroutine stellarvshalomassrelationleauthaud2012reduce
function volumefunction1dconstructorinternal
function volumefunction1dloglikelihood
function nbodymassconstructorinternal
function normalconstructorinternal
function grvtnllnsngoperatescalar
function randomerroroperatedistribution
subroutine sequencedeepcopy
function spinnbodyerrorsconstructorinternal
function spinnbodyerrorsoperatescalar
function luminosityfunctionconstructorinternal
function luminosityfunctiongunawardhana2013sdssconstructorinternal
function luminosityfunctionmonterodorta2009sdssconstructorinternal
function massfunctionstellarconstructorinternal
function massfunctionstellarprimusconstructorinternal
function massfunctionstellarultravistaconstructorinternal
function massfunctionstellarzfourgeconstructorinternal
function massmetallicityblanc2017constructorinternal
function meanfunction1dloglikelihood
subroutine multideepcopy
function himassoperate
function csmlgyangulardistanceoperate
function csmlgyluminositydistanceoperate
function squareoperate
function scatterfunction1dconstructorparameters
subroutine scatterfunction1dreduce
function stellarvshalomassrelationleauthaud2012constructorinternal
function output_analysis_output_-weight_survey_volume
function volumefunction1dconstructorparameters
subroutine volumefunction1dreduce
function csmlgyvolumeoperate
function propertyconstructorinternal

subroutine sequencedeepcopy
function galacticus_build_string
function squareconstructorinternal
function squareisinlightcone
subroutine squarereplicants
subroutine geometrymanglebuild
subroutine windowread
function bernardi2013sdssdistancemaximum
function caputi2011ukidssudsconstructorinternal
subroutine caputi2011ukidssudsrandomsinitialize
function caputi2011ukidssudsvolumemaximum
function davidzon2013vipersdistancemaximum
function gunawardhana2013sdssconstructorinternal
function hearin2014sdssconstructorinternal
function kelvin2014gamaneardistanceminimum
subroutine liwhite2009sdssrandomsinitialize
function localgroupdesdistancemaximum
function martin2010alfalfadistancemaximum
function martin2010alfalfasolidangle
function monterodorta2009sdssdistancemaximum
function moustakas2013primusconstructorinternal
function moustakas2013primusvolumemaximum
function muzzin2013ultravistadistancemaximum
function tomczak2014zfourgeconstructorinternal
function tomczak2014zfourgevolumemaximum
subroutine combineddeepcopy
function combinedsolidangle
subroutine fullskywindowfunctions
subroutine galacticus_build_output
subroutine galacticus_version_output
function squareconstructorparameters
function squareposition
function geometrymangleangularpower
function geometrymanglesolidangle
function baldry2012gamadistancemaximum
subroutine bernardi2013sdssmanglefiles
function caputi2011ukidssudsdistancemaximum
function caputi2011ukidssudssolidangle
function davidzon2013vipersconstructorinternal
function davidzon2013vipersvolumemaximum
function gunawardhana2013sdssdistancemaximum
function kelvin2014gamaneardistancemaximum
function liwhite2009sdssdistancemaximum
function liwhite2009sdsssolidangle
function localgroupsdssdistancemaximum
subroutine martin2010alfalfarandomsinitialize
function monterodorta2009sdssconstructorinternal
function monterodorta2009sdssdistanceminimum
function moustakas2013primusdistancemaximum
function muzzin2013ultravistaconstructorinternal
function muzzin2013ultravistavolumemaximum
function tomczak2014zfourgedistancemaximum
function combinedangularpower
function combinedpointincluded
subroutine combinedwindowfunctions
function mangleangularpower

function manglepointincluded
function manglesolidangle
subroutine manglewindowfunctions
function randompointincluded
function randompointsangularpower
subroutine randompointswindowfunctions
function behroozi2010massfunction
function triaxialityconstructorinternal
subroutine triaxialitymodify
subroutine halo_model_projected_correlation
function emissionlinewavelength
function enzohydrostaticconstructorinternal
function patejloeb2015constructorinternal
function ricotti2000constructorinternal
function betaprofileconstructorparameters
function growingconstructorinternal
function velocitymaximumscalingconstructorinternal
function font2008radiusstripped
subroutine filter_response_load
function filter_vega_offset
function filter_wavelength_effective
subroutine interface_camb_initialize
subroutine interface_camb_transfer_function
subroutine interface_cloudy_initialize
subroutine interface_cloudy_cie_tabulate
subroutine interface_fsps_initialize
subroutine interface_fsps_ssps_tabulate
subroutine localgroupdbgetproperty
subroutine localgroupdbselect
function vector3dcomparisonunimplemented
subroutine interface_recfast_initialize
function gnedin2000coefficientsearlyepoch
subroutine gnedin2000tabulate
function recfastconstructorinternal
subroutine filereaddata
function instantreionizationigmconstructorinternal
subroutine internalstateset
function miyamotonagaiconstructorinternal
function miyamotonagaisurfacedensityradialmoment
function exponentialdiskconstructorinternal
function exponentialdiskdensity
function exponentialdisksurfacedensityradialmoment
function hernquistconstructorinternal
function hernquistdensityradialmoment
function nfwconstructorinternal
function sersicconstructorinternal
function sersicdensityradialmoment
function betaprofileconstructorinternal
function radialmomenttwothirds
function fastexponentiatorexponentiate
function inverse_gamma_function_incomplete_complementary
function hypergeometric_2f1
function matrixmatrixmultiply
function matrix_copy_upper_to_lower_triangle
function parkinsoncolehellyconstructorinternal
function parkinsoncolehellymassbranchrootderivative

function parkinsoncolehellyprobabilitybound
subroutine parkinsoncolehellyupperboundhypergeometrictabulate
function generalizedpressschechterfractionsubresolutionm
subroutine buildconstructmasses
function fixedmassconstructorparameters
subroutine readxmlread
subroutine sampleddistributionconstruct

subroutine unionconstruct
function cole2000constructorinternal
function indexnode
function fullyspecifiedconstructorinternal
subroutine readassignnamedproperties
subroutine readassignspinparameters
subroutine readbuilddescendentpointers

subroutine readbuildparentpointers

function readconstruct
subroutine readcreatenodeindices
subroutine readintertreemergetimeset
subroutine readtimeuntilmergingsubresolution
function importerunitconvert2d
function importerunitsareequal
subroutine sussingasciiopen
function sussingangularmomentaincludesubhalos
subroutine sussingload
subroutine sussingsubhalotrace
function sussingtreeweight
subroutine sussinghdf5load
subroutine sussinghdf5open

function galacticuscubelength
subroutine galacticusimport

subroutine galacticusopen
function galacticussubhalotracecount
subroutine mergertreestatefromfile

subroutine parkinsoncolehellysubresolutionhypergeometrictabulate
function generalizedpressschechterconstructorinternal
function generalizedpressschechtermassbranch
function buildconstructorparameters
subroutine readhdf5read
subroutine sampleddistributioncmf
function sampleddistributionconstructorparameters
subroutine cole2000build
function fullyspecifiedconstruct
function nodelookup
subroutine readassignmergers

subroutine readassignscaleradii
subroutine readassignsplitforestevents
subroutine readbuildisolatedparentpointers
subroutine readbuildsubhalomasshistories
function readconstructorinternal
subroutine readenforcesubhalostatus
subroutine readscanformergers
function importerunitconvert1d

function importerunitconvertscalar
subroutine sussingasciiload
subroutine sussingasciireadhalo
subroutine sussingimport

function sussingmassesincludesubhalos
subroutine sussingtreeindicesread
function sussingvalueisbad
function decodeunits
function galacticusangularmomentaincludesubhalos
subroutine galacticusforestindicesread
function galacticusmassesincludesubhalos
subroutine galacticussubhalotrace
function galacticustreeweight
subroutine mergertreestatestore

function nodearrayposition
subroutine historystore
subroutine historywrite
subroutine multideepcopy
subroutine recordevolutionoutput
subroutine recordevolutionstore
subroutine nonevolvingevolve
subroutine standardevolve
function standardtimeevolveto
function standardconstructorinternal
subroutine standardevolve
function standardodes
subroutine singlelevelhierarchyprocess
function augmentbuildtreefromnode
function augmentconstructorparameters
subroutine augmentextendnonoverlapnodes
subroutine augmentsortchildren
function augmenttreestatistics
function conditionalmfbinweights2d
function conditionalmfbinweights2dintegrand
function conditionalmfconstructorinternal
subroutine conditionalmfoperate
function exportconstructorinternal
function informationcontentconstructorinternal
function massaccretionhistoryconstructorinternal
subroutine massaccretionhistoryoperate
subroutine outputrootmassesoperate
function particulateconstructorinternal
subroutine particulateoperate
subroutine particulatetabulateenergydistribution
function prunehierarchyconstructorinternal
subroutine prunelightconevalidate
function prunenonessentialconstructorinternal
function regridtimesconstructorinternal
function regridtimesconstructorparameters
subroutine regridtimesoperate
subroutine sequencedeepcopy
subroutine analyzerreduce
subroutine multideepcopy
subroutine multireduce
subroutine standardoutput
subroutine standardpropertiescount
subroutine standardpropertynamesestablish
function allandformationnodesparameters
subroutine allandformationnodesprevious
function allnodesparameters
subroutine allnodesprevious
function allnodesbranchparameters
function isolatednodesparameters
function isolatednodesbranchparameters
function treeconstructionparameters
function fileconstructorinternal
function sedfitconstructorinternal
function sedfitevaluate
function galaxypopulationevaluate
function gaussianregressionconstructorinternal
function gaussianregressionevaluate
subroutine gaussianregressionfitvariogram
function halomassfunctionconstructorinternal
function halomassfunctionevaluate
function independentlikelihoodsconstructorparameters
function independentlikelihoodsevaluate

function independentlikelihoodssequantialevaluate
function massfunctionevaluate
function projectedcorrelationfunctionevaluate
function spindistributionevaluate
function inactivemap
function inactivepriormaximum
function inactivepriorsample
function inactiveunmap
function gadgethdf5import
function boundupper
subroutine rotationcurveoperate
subroutine sequencedeepcopy
subroutine velocitydispersionoperate
function lightconeextract
subroutine multideepcopy
function multielementcount
function multiextractinteger
function multiunitsinsi
function ratioextract
function satellitestatusconstructorinternal
subroutine ode_solve
subroutine latentintegrator
function integrate
function compositegausskronrod1devaluate
function compositetrapezoidalevaluate1d
function multivectorizedcompositegausskronrod1devaluate
function multivectorizedcompositetrapezoidal1devaluate
subroutine tolerancesetgeneric
function vectorizedcompositegausskronrod1devaluate
function vectorizedcompositetrapezoidalevaluate1d
function independentlikelihoodssequentialconstructorparameters
function likelihoodmassfunctiontimeintegrand
function spindistributionconstructorinternal
function inactivelogprior
function inactivepriorinvert
function inactivepriorminimum
function inactiverandomperturbation
function gadgetbinaryimport
function boundlower
subroutine meanpositionoperate
subroutine selfboundoperate
function velocitydispersionconstructorinternal
subroutine lightconeaddinstances
function lmnstyemssnlineconstructorinternal
function multidescriptions
function multiextractdouble
function multinames
function ratioconstructorinternal
function satelliteorbitalextremaconstructorinternal
subroutine fftlog
subroutine odeiv2_solve
function differentiatorderivative
function adaptivecompositetrapezoidalevaluate1d
subroutine compositegausskronrod1dinitialize
subroutine compositetrapezoidalinitialize1d
subroutine multivectorizedcompositegausskronrod1dinitialize
subroutine multivectorizedcompositetrapezoidal1dinitialize
subroutine tolerancessetgeneric
subroutine vectorizedcompositegausskronrod1dinitialize
subroutine vectorizedcompositetrapezoidalinitialize1d

function interpolate
function interpolate_derivative
function interpolate_locate
subroutine meshes_apply_point
subroutine nearestneighborsclose
function nearestneighborsconstructor
subroutine nearestneighborsdestructor
subroutine nearestneighborssearch
subroutine nearestneighborssearchfixedradius
function make_range
function root_finder_find
subroutine root_finder_range_expand
function search_array_for_closest
subroutine sort_topological
function abundances_atomic_index
subroutine abundances_builder
function abundances_get_metallicity
function abundances_names
subroutine abundances_set_metallicity
subroutine chemicals_builder
function chemicals_names
function chemical_database_get_index
subroutine chemical_structure_initialize
subroutine coordinates_null_from
function coordinates_null_to
subroutine debugstackpop
subroutine debugstackpushloc
subroutine debugstackpushstr
function history_add
subroutine history_append_epoch
subroutine history_append_history
subroutine history_builder
subroutine history_create
subroutine history_deserialize
subroutine history_extend
subroutine history_increment
subroutine history_interpolated_increment
subroutine history_long_integer_append_epoch
subroutine history_long_integer_append_history
subroutine history_long_integer_builder
subroutine history_long_integer_create
subroutine history_long_integer_trim
function history_subtract
subroutine history_trim
subroutine kepler_orbits_assert_is_defined
subroutine kepler_orbits_builder
function kepler_orbits_epsilon
function kepler_orbits_host_mass
function kepler_orbits_phi
subroutine kepler_orbits_propagate
function kepler_orbits_radius
function kepler_orbits_specific_reduced_mass
function kepler_orbits_theta
function kepler_orbits_velocity_radial
function kepler_orbits_velocity_scale
function kepler_orbits_velocity_tangential
subroutine merger_tree_data_construct_particle_indices
subroutine merger_tree_data_set_subhalo_masses
subroutine merger_tree_data_structure_add_metadata
subroutine merger_tree_data_structure_convert_property_units
subroutine merger_tree_data_structure_export
subroutine merger_tree_data_structure_export_galacticus
subroutine merger_tree_data_structure_export_irate
subroutine merger_tree_data_structure_read_ascii
subroutine merger_tree_data_structure_read_particles_ascii
subroutine merger_tree_data_structure_set_conversion_factor

subroutine merger_tree_data_structure_-set_property_double
subroutine merger_tree_data_structure_-set_property_integer8
subroutine merger_tree_data_structure_-set_units
subroutine merger_tree_data_validate_-trees
module galacticus_nodes
subroutine agestatisticsdiskintegratedsfrrate
subroutine agestatisticsdisktimeweightedintegratedsfrrate
subroutine agestatisticsspheroidintegratedsfrrate
subroutine agestatisticsspheroidtimeweightedintegratedsfrrate
function agestatisticsstandarddiskintegratedsfrrateget
function agestatisticsstandarddisktimeweightedintegratedsfrrateget
function agestatisticsstandardspheroidintegratedsfrrateget
function agestatisticsstandardspheroidtimeweightedintegratedsfrrateget
function basicnonevolvingtimerateget
function basicstandardextendedmassbertschingerrateget
function basicstandardextendedradiusturnaroundrateget
function basicstandardmassrateget
function basicstandardtimerateget
function blackholenoncentralradialpositionrateget
function blackholesimplemassrateget
function blackholestandardmassrateget
function blackholestandardspinrateget
function darkmatterprofilescalepresetscalerateget
function darkmatterprofilescalescalerateget
function darkmatterprofilescaleshapeshaperateget
subroutine diskabundancesgasrate
subroutine diskangularmomentumrate
subroutine diskmassgasrate
subroutine diskstandardabundancesgasrategeneric
function diskstandardabundancesgasrateget
function diskstandardabundancesstellarrateget
subroutine diskstandardangularmomentumrategeneric
function diskstandardangularmomentumrateget
function diskstandardfractionmassretainedrateget
function diskstandardluminositiesstellarrateget
subroutine diskstandardmassgasrategeneric
function diskstandardmassgasrateget
function diskstandardmassstellarformedrateget
function diskstandardmassstellarrateget
function diskstandardstarformationhistoryrateget
function diskstandardstellarpropertieshistoryrateget
subroutine diskverysimpleabundancesgasrategeneric
function diskverysimpleabundancesgasrateget
function diskverysimpleabundancesstellarrateget
function diskverysimpleluminositiesstellarrateget
subroutine diskverysimplemassgasrategeneric
function diskverysimplemassgasrateget
function diskverysimplemassstellarrateget

function diskverysimplestellarpropertieshistoryrateget
subroutine hothaloabundancescoldrate
subroutine hothaloabundancesrate
subroutine hothaloangularmomentumcoldrate
subroutine hothaloangularmomentumrate
subroutine hothalochemicalsrate
function hothalocoldmodeabundancescoldrateget
function hothalocoldmodeangularmomentumcoldrateget
function hothalocoldmodemasscoldrateget
function hothalocoldmodeouterradiusgrowthrate
subroutine hothalocoldmodeoutflowreturn
subroutine hothalomasscoldrate
subroutine hothalomassrate
function hothalooutflowtrackingouterradiusgrowthrate
subroutine hothalooutflowtrackingoutflowreturn
function hothalooutflowtrackingtrackedoutflowabundancesrateget
function hothalooutflowtrackingtrackedoutflowmassrateget
function hothalostandardabundancesrateget
function hothalostandardangularmomentumrateget
function hothalostandardchemicalsrateget
function hothalostandardmassrateget
function hothalostandardouterradiusgrowthrate
function hothalostandardouterradiusrateget
function hothalostandardoutflowedabundancesrateget
function hothalostandardoutflowedangularmomentumrateget
function hothalostandardoutflowedmassrateget
subroutine hothalostandardoutflowreturn
function hothalostandardstrippedabundancesrateget
function hothalostandardstrippedmassrateget
function hothalostandardunaccretedmassrateget
subroutine hothalounaccretedmassrate
function hothaloverysimpleabundancesrateget
function hothaloverysimpledelayedoutflowedabundancesrateget
function hothaloverysimpledelayedoutflowedmassrateget
function hothaloverysimplemassrateget
function hothaloverysimpleunaccretedmassrateget
subroutine interoutputdiskstarformationraterate
subroutine interoutputspheroidstarformationraterate
function interoutputstandarddiskstarformationrateget
function interoutputstandardspheroidstarformationrateget
function massflowstatisticsstandardcooledmassrateget
function nbodygenericaddintegerproperty
function nbodygenericaddrealproperty
subroutine nbodygenericsetintegerproperty
subroutine nbodygenericsetrealproperty
subroutine node_component_disk_standard_attach_pipes

function node_component_disk_standard_half_mass_radius
function node_component_disk_very_simple_enclosed_mass
function node_component_spheroid_standard_half_mass_radius
subroutine nodecomponentagestatisticsbuilder
subroutine nodecomponentagestatisticsinitialize
subroutine nodecomponentagestatisticsnullbuilder
subroutine nodecomponentagestatisticsstandardbuilder
subroutine nodecomponentbasiccreate
subroutine nodecomponentbasicinitialize
subroutine nodecomponentbasicnonevolvingbuilder
subroutine nodecomponentbasicremove
subroutine nodecomponentbasicstandardextendedbuilder
subroutine nodecomponentblackholebuilder
subroutine nodecomponentblackholeinitialize
subroutine nodecomponentblackholenoncentralbuilder
subroutine nodecomponentblackholeremove
subroutine nodecomponentblackholestandardbuilder
subroutine nodecomponentdarkmatterprofilecreate
subroutine nodecomponentdarkmatterprofilemove
subroutine nodecomponentdarkmatterprofileremove
subroutine nodecomponentdarkmatterprofilescalepresetbuilder
subroutine nodecomponentdiskbuilder
subroutine nodecomponentdiskinitialize
subroutine nodecomponentdisknullbuilder
subroutine nodecomponentdiskstandardbuilder
subroutine node_component_disk_very_simple_attach_pipe
subroutine node_component_hot_halo_standard_mass_removal_rate
function node_component_spheroid_very_simple_enclosed_mass
subroutine nodecomponentagestatisticscreate
subroutine nodecomponentagestatisticsmove
subroutine nodecomponentagestatisticsremove
subroutine nodecomponentbasicbuilder
subroutine nodecomponentbasicextendedtrackingbuilder
subroutine nodecomponentbasicmove
subroutine nodecomponentbasicnullbuilder
subroutine nodecomponentbasicstandardbuilder
subroutine nodecomponentbasicstandardtrackingbuilder
subroutine nodecomponentblackholecreate
subroutine nodecomponentblackholemove
subroutine nodecomponentblackholenullbuilder
subroutine nodecomponentblackholesimplebuilder
subroutine nodecomponentdarkmatterprofilebuilder
subroutine nodecomponentdarkmatterprofileinitialize
subroutine nodecomponentdarkmatterprofilenullbuilder
subroutine nodecomponentdarkmatterprofilescalebuilder
subroutine nodecomponentdarkmatterprofilescaleshapebuilder
subroutine nodecomponentdiskcreate
subroutine nodecomponentdiskmove
subroutine nodecomponentdiskremove
subroutine nodecomponentdiskverysimplebuilder

subroutine nodecomponentdiskverysimplesizebuilder
subroutine nodecomponentdynamicsstatisticsbarsbuilder
subroutine nodecomponentdynamicsstatisticsbuilder
subroutine nodecomponentdynamicsstatisticscreate
subroutine nodecomponentdynamicsstatisticsinitialize
subroutine nodecomponentdynamicsstatisticsmove
subroutine nodecomponentdynamicsstatisticsnullbuilder
subroutine nodecomponentdynamicsstatisticsremove
subroutine nodecomponentformationtimebuilder
subroutine nodecomponentformationtimecole2000builder
subroutine nodecomponentformationtimecreate
subroutine nodecomponentformationtimeinitialize
subroutine nodecomponentformationtimemassfractionbuilder
subroutine nodecomponentformationtimemove
subroutine nodecomponentformationtimenullbuilder
subroutine nodecomponentformationtimeremove
subroutine nodecomponentgeterror
subroutine nodecomponenthosthistorybuilder
subroutine nodecomponenthosthistorycreate
subroutine nodecomponenthosthistoryinitialize
subroutine nodecomponenthosthistorymove
subroutine nodecomponenthosthistorynullbuilder
subroutine nodecomponenthosthistoryremove
subroutine nodecomponenthosthistorystandardbuilder
subroutine nodecomponenthothalobuilder
subroutine nodecomponenthothalocoldmodebuilder
subroutine nodecomponenthothalocreate
subroutine nodecomponenthothaloinitialize
subroutine nodecomponenthothalomove
subroutine nodecomponenthothalonullbuilder
subroutine nodecomponenthothalooutflowtrackingbuilder
subroutine nodecomponenthothaloremove
subroutine nodecomponenthothalostandardbuilder
subroutine nodecomponenthothaloverysimplebuilder
subroutine nodecomponenthothaloverysimpledelayedbuilder
subroutine nodecomponentindicesbuilder
subroutine nodecomponentindicescreate
subroutine nodecomponentindicesinitialize
subroutine nodecomponentindicesmove
subroutine nodecomponentindicesnullbuilder
subroutine nodecomponentindicesremove
subroutine nodecomponentindicesstandardbuilder
subroutine nodecomponentinteroutputbuilder
subroutine nodecomponentinteroutputcreate

subroutine nodecomponentinteroutputinitialize
subroutine nodecomponentinteroutputmove
subroutine nodecomponentinteroutputnullbuilder
subroutine nodecomponentinteroutputremove
subroutine nodecomponentinteroutputstandardbuilder
subroutine nodecomponentmassflowstatisticsbuilder
subroutine nodecomponentmassflowstatisticscreate
subroutine nodecomponentmassflowstatisticsinitialize
subroutine nodecomponentmassflowstatisticsmove
subroutine nodecomponentmassflowstatisticsnullbuilder
subroutine nodecomponentmassflowstatisticsremove
subroutine nodecomponentmassflowstatisticsstandardbuilder
subroutine nodecomponentmergingstatisticsbuilder
subroutine nodecomponentmergingstatisticscreate
subroutine nodecomponentmergingstatisticsinitialize
subroutine nodecomponentmergingstatisticsmajorbuilder
subroutine nodecomponentmergingstatisticsmove
subroutine nodecomponentmergingstatisticsnullbuilder
subroutine nodecomponentmergingstatisticsrecentbuilder
subroutine nodecomponentmergingstatisticsremove
subroutine nodecomponentmergingstatisticsstandardbuilder
subroutine nodecomponentnbodybuilder
subroutine nodecomponentnbodycreate
subroutine nodecomponentnbodygenericbuilder
subroutine nodecomponentnbodyinitialize
subroutine nodecomponentnbodymove
subroutine nodecomponentnbodynullbuilder
subroutine nodecomponentnbodyremove
subroutine nodecomponentpositionbuilder
subroutine nodecomponentpositioncreate
subroutine nodecomponentpositioninitialize
subroutine nodecomponentpositionmove
subroutine nodecomponentpositionnullbuilder
subroutine nodecomponentpositionpresetbuilder
subroutine nodecomponentpositionpresetorphansbuilder
subroutine nodecomponentpositionremove
subroutine nodecomponentpositiontracedarkmatterbuilder
subroutine nodecomponentsatellitebuilder
subroutine nodecomponentsatellitecreate
subroutine nodecomponentsatelliteinitialize
subroutine nodecomponentsatellitemove
subroutine nodecomponentsatellitenullbuilder
subroutine nodecomponentsatelliteorbitingbuilder
subroutine nodecomponentsatellitepresetbuilder
subroutine nodecomponentsatelliteremove
subroutine nodecomponentsatellitestandardbuilder

subroutine nodecomponentsatelliteverysimplebuilder
subroutine nodecomponentspheroidbuilder
subroutine nodecomponentspheroidcreate
subroutine nodecomponentspheroidinitialize
subroutine nodecomponentspheroidmove
subroutine nodecomponentspheroidnullbuilder
subroutine nodecomponentspheroidremove
subroutine nodecomponentspheroidstandardbuilder
subroutine nodecomponentspheroidverysimplebuilder
subroutine nodecomponentspinbuilder
subroutine nodecomponentspincreate
subroutine nodecomponentspininitialize
subroutine nodecomponentspinmove
subroutine nodecomponentspinnullbuilder
subroutine nodecomponentspinpreset3dbuilder
subroutine nodecomponentspinpresetbuilder
subroutine nodecomponentspinrandombuilder
subroutine nodecomponentspinremove
subroutine nodecomponentspinvitvitskabuilder
subroutine nullagestatisticsscaleinoutdouble0
subroutine nullagestatisticssetinoutdouble0
subroutine nullbasicscaleinoutdouble0
subroutine nullbasicsetinoutdouble0
subroutine nullblackholescaleinoutdouble0
subroutine nullblackholesetinoutdouble0
subroutine nulldarkmatterprofilescaleinoutdouble0
subroutine nulldarkmatterprofilesetinoutdouble0
subroutine nulldarkmatterprofilesetinoutlogical0
subroutine nulldiskscaleinoutabundances0
subroutine nulldiskscaleinoutdouble0
subroutine nulldiskscaleinouthistory0
subroutine nulldiskscaleinoutstellarluminosities0
subroutine nulldisksetinoutabundances0
subroutine nulldisksetinoutdouble0
subroutine nulldisksetinouthistory0
subroutine nulldisksetinoutlogical0
subroutine nulldisksetinoutstellarluminosities0
subroutine nulldynamicsstatisticssetinoutdouble1
subroutine nullformationtimesetinoutdouble0
subroutine nullhosthistorysetinoutdouble0
subroutine nullhothaloscaleinoutabundances0
subroutine nullhothaloscaleinoutchemicalabundances0
subroutine nullhothaloscaleinoutdouble0
subroutine nullhothalosetinoutabundances0
subroutine nullhothalosetinoutchemicalabundances0
subroutine nullhothalosetinoutdouble0
subroutine nullhothalosetinoutlogical0
subroutine nullindicessetinoutlonginteger0

subroutine nullinteroutputscaleinoutdouble0
subroutine nullinteroutputsetinoutdouble0
subroutine nullmassflowstatisticsscaleinoutdouble0
subroutine nullmassflowstatisticssetinoutdouble0
subroutine nullmergingstatisticssetinoutdouble0
subroutine nullmergingstatisticssetinoutdouble1
subroutine nullmergingstatisticssetinoutinteger0
subroutine nullmergingstatisticssetinoutinteger1
subroutine nullnbodysetinoutdouble1
subroutine nullnbodysetinoutlonginteger1
subroutine nullpositionsetinoutdouble0
subroutine nullpositionsetinoutdouble1
subroutine nullpositionsetinouthistory0
subroutine nullsatellitescaleinoutdouble0
subroutine nullsatellitescaleinoutdouble1
subroutine nullsatellitescaleinouttensorrank2dimension3symmetric0
subroutine nullsatellitesetinoutdouble0
subroutine nullsatellitesetinoutdouble1
subroutine nullsatellitesetinouthistory0
subroutine nullsatellitesetinoutkeplerorbit0
subroutine nullsatellitesetinoutlogical0
subroutine nullsatellitesetinoutlongintegerhistory0
subroutine nullsatellitesetinouttensorrank2dimension3symmetric0
subroutine nullspheroidscaleinoutabundances0
subroutine nullspheroidscaleinoutdouble0
subroutine nullspheroidscaleinouthistory0
subroutine nullspheroidscaleinoutstellarluminosities0
subroutine nullspheroidsetinoutabundances0
subroutine nullspheroidsetinoutdouble0
subroutine nullspheroidsetinouthistory0
subroutine nullspheroidsetinoutlogical0
subroutine nullspheroidsetinoutstellarluminosities0
subroutine nullspinscaleinoutdouble0
subroutine nullspinscaleinoutdouble1
subroutine nullspinsetinoutdouble0
subroutine nullspinsetinoutdouble1
function satelliteorbitingboundmassrateget
function satelliteorbitingpositionrateget
function satelliteorbitingtidalheatingnormalizedrateget
function satelliteorbitingtidaltensorpathintegratedrateget
function satelliteorbitingvelocityrateget
function satellitestandardboundmassrateget
function satellitestandardmergetimerateget
function satelliteverysimplemergetimerateget
subroutine spheroidabundancesgasrate
subroutine spheroidabundancesstellarrate
subroutine spheroidangularmomentumrate
subroutine spheroidluminositiesstellarrate
subroutine spheroidmassgasrate
subroutine spheroidmassstellarrate
function spheroidstandardabundancesgasrateget
function spheroidstandardabundancesstellarrateget

function spheroidstandardangularmomentumrateget
function spheroidstandardluminositiesstellarrateget
function spheroidstandardmassgasrateget
function spheroidstandardmassstellarformedrateget
function spheroidstandardmassstellarrateget
function spheroidstandardstarformationhistoryrateget
function spheroidstandardstellarpropertieshistoryrateget
subroutine spheroidstarformationhistoryrate
subroutine spheroidstellarpropertieshistoryrate
function spheroidverysimpleabundancesgasrateget
function spheroidverysimpleabundancesstellarrateget
function spheroidverysimpleluminositiesstellarrateget
function spheroidverysimplemassgasrateget
function spheroidverysimplemassstellarrateget
function spheroidverysimplestellarpropertieshistoryrateget
function spinpreset3dspinvectorrateget
function spinpresetspinrateget
function spinrandomspinrateget
function spinvitvitskaspinrateget
function spinvitvitskaspinvectorrateget
function tree_node_get_earliest_progenitor
function tree_node_index
function tree_node_is_on_main_branch
function tree_node_is_primary_progenitor
function tree_node_is_primary_progenitor_of_index
function tree_node_is_primary_progenitor_of_node
function tree_node_is_progenitor_of_index
function tree_node_is_progenitor_of_node
function tree_node_is_satellite
subroutine tree_node_remove_from_host
subroutine tree_node_remove_from_mergee
function tree_node_unique_id
function treenodeagestatisticscount
function treenodebasiccount
function treenodeblackholecount
function treenodedarkmatterprofilecount
function treenodediskcount
function treenodedynamicsstatisticscount
function treenodeformationtimecount
function treenodehosthistorycount
function treenodehothalocount
function treenodeindicescount
subroutine treenodeinitialize
function treenodeinteroutputcount
function treenodemapdouble0
function treenodemassflowstatisticscount
function treenodemergingstatisticscount
function treenodenbodycount
function treenodepositioncount
function treenodesatellitecount
function treenodespheroidcount
function treenodespincount
subroutine node_component_age_statistics_standard_satellite_merging
subroutine node_component_basic_standard_extended_promote
subroutine node_component_basic_extended_tracking_promote
subroutine node_component_basic_non_evolving_promote

subroutine node_component_basic_standard_promote
subroutine node_component_basic_standard_tracking_promote
subroutine node_component_dark_matter_profile_scale_promote
subroutine node_component_dark_matter_profile_scale_tree_initialize
subroutine node_component_dark_matter_profile_scale_preset_promote
subroutine node_component_dark_matter_profile_scale_shape_promote
subroutine node_component_disk_standard_initialize
subroutine node_component_disk_standard_post_step
subroutine node_component_disk_standard_rate_compute
subroutine node_component_disk_standard_satellite_merging
subroutine node_component_disk_standard_state_retrieve
subroutine node_component_disk_standard_thread_initialize
subroutine node_component_disk_very_simple_analytic_solver
subroutine node_component_disk_very_simple_satellite_merging
subroutine node_component_dynamics_statistics_bars_rate_compute
subroutine node_component_formation_time_mass_fraction_node_promotion
subroutine node_component_hot_halo_cold_mode_outflow_return
subroutine node_component_hot_halo_cold_mode_push_to_cooling_pipes
subroutine node_component_hot_halo_standard_heat_source
subroutine node_component_hot_halo_standard_initialize
subroutine node_component_hot_halo_standard_mass_sink
subroutine node_component_hot_halo_standard_outflow_return
subroutine node_component_hot_halo_standard_push_to_cooling_pipes
subroutine node_component_hot_halo_standard_thread_initialize
subroutine node_component_inter_output_standard_satellite_merging
subroutine node_component_merging_statistics_recent_initialize
subroutine node_component_merging_statistics_recent_node_merger
subroutine node_component_nbody_generic_set_integer_property
subroutine node_component_nbody_generic_set_real_property
subroutine node_component_position_trace_dark_matter_initialize
subroutine node_component_position_trace_dark_matter_update
subroutine node_component_satellite_orbiting_bound_mass_initialize
subroutine node_component_satellite_orbiting_initialize
subroutine node_component_spheroid_standard_energy_gas_input_rate
subroutine node_component_spheroid_standard_initialize
subroutine node_component_spheroid_standard_mass_gas_sink_rate
subroutine node_component_spheroid_standard_post_step
subroutine node_component_spheroid_standard_rate_compute
subroutine node_component_spheroid_standard_satellite_merging
subroutine node_component_spheroid_standard_star_formation_history_rate
subroutine node_component_spheroid_standard_state_retrieve
subroutine node_component_spheroid_standard_stellar_prprts_history_rate
subroutine node_component_spheroid_standard_thread_initialize
subroutine node_component_spheroid_very_simple_satellite_merging

subroutine node_component_spin_vitvitska_promote
subroutine node_component_spin_preset_promote
subroutine node_component_spin_preset3d_promote
subroutine node_component_spin_random_promote
subroutine stellar_luminosities_builder
function stellar_luminosities_index_from_name
function stellar_luminosities_index_from_properties
subroutine stellar_luminosities_initialize
function stellar_luminosities_is_output
function stellar_luminosities_luminosity
function stellar_luminosities_name
function table1d_find_effective_x
function table1d_integration_weights
subroutine table_1d_reverse
subroutine table_2d_linlinlin_create
subroutine table_2d_linlinlin_populate
subroutine table_2d_linlinlin_populate_single
function table_2dlogloglin_interpolate_gradient
subroutine table_2dlogloglin_populate
subroutine table_2dlogloglin_populate_single
function table_2dlogloglin_size
subroutine table_generic_1d_create
subroutine table_generic_1d_populate
subroutine table_generic_1d_populate_single
subroutine table_linear_1d_create
subroutine table_linear_1d_populate
subroutine table_linear_1d_populate_single
subroutine table_linear_cspline_1d_create
subroutine table_linear_cspline_1d_populate
subroutine table_linear_cspline_1d_populate_single
function table_linear_cspline_integration_weights
subroutine table_linear_monotone_cspline_1d_create
subroutine table_linear_monotone_cspline_1d_populate
subroutine table_linear_monotone_cspline_1d_populate_single
function table_linear_monotone_cspline_integration_weights
subroutine table_logarithmic_1d_reverse
function table_logarithmic_integration_weights
subroutine table_nonuniform_linear_logarithmic_1d_populate
subroutine table_nonuniform_linear_logarithmic_1d_populate_single
function table_nonuniform_linear_logarithmic_integration_weights
subroutine tensor_r2_d3_sym_assign_to
subroutine tensor_r2_d3_sym_builder
subroutine tensor_r2_d3_sym_from_matrix
function listindex
function gelmanrubinconstructorinternal
function differentialevolutionconstructorparameters
subroutine differentialevolutionposterior
subroutine differentialevolutionsimulate
function particleswarmconstructorparameters
subroutine particleswarmposterior
subroutine particleswarmsimulate
function stochasticdifferentialevolutionacceptproposal

function historymean
subroutine resumeinitialize
function simplemean
function simplevariance
function intergalacticbackgroundfileconstructorinternal
subroutine intergalacticbackgroundinternaluniverseplanetvel
subroutine summationdeepcopy
subroutine cole2000get
subroutine simpleget
subroutine standardget
subroutine covington2008get

function jiang2008constructorparameters
function benson2005orbit

function jiang2014orbit

function wetzel2010orbit

function wetzel2010velocitytangentialvectormean
function fixedconstructorinternal
function fixedvelocitytangentialvectormean
function spincorrelatedorbit

subroutine satellite_orbit_extremum_phase_space_coordinates
function fixedconstructorparameters

function powerlawconstructorinternal

function fixedconstructorparameters
function powerlawconstructorparameters

subroutine insitumake
function metallicitysplitconstructorinternal
function blitz2006constructorinternal

function krumholz2009constructorparameters

function historyvariance
subroutine switchedinitialize
subroutine simplerestore
function correlationlengthstop
function intergalacticbackgroundinternalflux
function intergalacticbackgroundinternalupdate
function cole2000constructorparameters
function simpleconstructorparameters
function standardconstructorparameters
subroutine cole2000get
function boylankolchin2008timeuntilmerging
function jiang2008timeuntilmerging
function benson2005velocitytangentialvectormean
function jiang2014velocitytangentialvectormean
function wetzel2010velocitytangentialmagnitudemean
function wetzel2010velocitytotalrootmeansquared
function fixedorbit
function isotropicvelocitytangentialvectormean
function equivalent_circular_orbit_solver
function creasey2012constructorparameters
function haloscalingconstructorparameters
function vlctymxsclngconstructorparameters
function powerlawconstructorinternal
function vlctymxsclngconstructorparameters
subroutine insitusatellitemerger
subroutine metallicitysplitmake

function kennicuttschmidtconstructorparameters
function extendedschmidtconstructorparameters

function dynamicaltimeconstructorinternal
function dynamicaltimeconstructorinternal
function nbodyhalomasserrortrenti2010internal
function cauchyinverse
function gammaconstructorinternal
function gammainverse
function studenttinverse
function betainverse
function binomialconstructorinternal
function binomialcumulative
function binomialinverse
function binomialmass
function negativebinomialconstructorinternal
function negativebinomialcumulative
function negativebinomialinverse
function negativebinomialmass
function negativebinomialmasslogarithmic
function loguniforminverse
function negativeexponentialinverse
function normalinverse
function normalmaximum
function normalminimum
function normalstandardinverse
function peakbackgroundinverse
function peakbackgroundmaximum
function peakbackgroundminimum
function uniforminverse
subroutine statistics_points_correlation
subroutine statistics_points_power_spectrum
subroutine fileread
function hegerwoosley2002constructorinternal
function nagashima2005constructorinternal
function fileconstructorinternal
subroutine stellarpopulationuniqueidassign
function piecewisepowerlawconstructorinternal
function stellar_population_luminosity
subroutine stellar_population_luminosity_tabulate
subroutine stellar_population_luminosity_track
function diskspheroidselect
function gordon2003constructorinternal
function prevotbouchetconstructorinternal
function prevotbouchetconstructorparameters
function wittgordon2003constructorinternal
function fileconstructorinternal
function fileluminosity
subroutine filereadfile
function lookupbuild
function lookupconstructorinternal
function lookupconstructorparameters
subroutine sequencedeepcopy
function standardinterpolate
subroutine filteredpowerretabulate
function variancepeakbackgroundsplitpowernormalization
function kitayamasuto1996constructorinternal
function kitayamasuto1996constructorparameters
function sphericalcollapsematterdeconstructorinternal
function sphericalcollapsematterdeconstructorparameters
function sphericalcollapsematterlambdaconstructorinternal

function sphericalcollapsematterlambdavalue
function barkana2001wdmconstructorinternal
function barkana2001wdmgradientmass
function barkana2001wdmvalue
function remapshethmotormenconstructorinternal
function remapscaleconstructorinternal
function zhanghuirate
function zhanghuiratenoncrossing
subroutine takahashi2011lensingdistributionconstruct
subroutine lognormaloverdensitylinearset
function lognormaloverdensitynonlinear
subroutine normaloverdensitylinearset
subroutine uniformoverdensitylinearset
function uniformpdf
function pressschechterdifferential
function rodriguezpuebla2016constructorinternal
function tinker2008constructorinternal
function fofbiasdifferential
function simplelogarithmicderivativeexpansionfactor
function cosmicemuconstructorinternal
function cosmicemuvalue
function peacockdodds1996value
subroutine make_table
subroutine make_table
subroutine restore_table
subroutine spherical_collape_matter_-lambda_delta_virial_tabulate
subroutine spherical_collapse_matter_-lambda_critical_overdensity_tabulate
function bbksconstructorinternal
function bbkshalfmodemass
function bbkswdmconstructorinternal
function bbkswdmhalfmodemass
function bode2001constructorinternal
function bode2001constructorparameters
function bode2001halfmodemass
function cambconstructorinternal
function eisensteinhu1999constructorinternal
function eisensteinhu1999halfmodemass
function filehalfmodemass
subroutine filereadfile
function identityhalfmodemass
function gammaincomplete
function bryannorman1998constructorinternal
function bryannorman1998densitycontrast
function bryannorman1998densitycontrastrateofchange
function kitayamasuto1996constructorinternal
function fixedconstructorinternal
function percolationconstructorinternal
subroutine percolationcopytable
subroutine percolationdeepcopy
function percolationdensitycontrast
function percolationdensitycontrastrateofchange
subroutine percolationdestructor
subroutine percolationfindparent
subroutine percolationtabulate
subroutine percolationobjectsdeepcopy
function virial_density_contrast_-percolation_solver
function sphericalcollapsematterderivativeofdeturnaroundoverviriali
function sphericalcollapsematterlambdadensitycontrast
function sphericalcollapsematterlambdadensitycontrastrateofchange
subroutine system_command_char

subroutine system_limits_set
subroutine localgroupdatabaseperform
subroutine agnspectrahopkins2008buildfileperform
subroutine buildtoolcambperform
subroutine buildtoolcloudyperform
subroutine buildtoolfspsperform
subroutine buildtoolrecfastperform
subroutine catalogprojectedcorrelationfunctionperform
function conditionalmassfunctionconstructorinternal
function conditionalmassfunctionconstructorparameters
subroutine conditionalmassfunctionperform
function evolveforestsconstructorparameters
subroutine evolveforestsperform
subroutine evolveforestssuspendtree
subroutine tasks_evolve_forest_construct
subroutine tasks_evolve_forest_destruct
subroutine tasks_evolve_forest_perform
function cyclicforestnumber
function fcfsforestnumber
function strideconstructorinternal
subroutine excursionsetsperform
subroutine halomassfunctionperform
subroutine halomodelprojectedcorrelationfunctionperform
subroutine halomodelgenerateperform
subroutine halospindistributionperform
subroutine intergalacticmediumstateperform
subroutine massfunctioncovarianceperform
subroutine mergertreefilebuilderperform
subroutine multideepcopy
subroutine multiperform
subroutine nbodyanalyzeperform
subroutine posteriorsampleperform
subroutine powerspectraperform
subroutine reportperform
program test_diemerkravtsov2014_concentration
program tests_io_xml
program test_integration2
program test_mass_distributions
program test_nodes
program test_root_finding
program test_sort_topological
function intergalacticmediumstateevolveconstructorinternal
function intergalacticmediumstateevolveupdate
subroutine io_hdf5_assert_attribute_type
subroutine io_hdf5_assert_dataset_type
subroutine io_hdf5_assert_in_critical
function io_hdf5_character_types
subroutine io_hdf5_close
subroutine io_hdf5_copy
subroutine io_hdf5_create_reference_scalar_to_1d
subroutine io_hdf5_create_reference_scalar_to_2d
subroutine io_hdf5_create_reference_scalar_to_3d
subroutine io_hdf5_create_reference_scalar_to_4d
subroutine io_hdf5_create_reference_scalar_to_5d
function io_hdf5_dataset_rank
function io_hdf5_dataset_size
function io_hdf5_datasets
subroutine io_hdf5_deep_copy
subroutine io_hdf5_flush
function io_hdf5_has_attribute
function io_hdf5_has_dataset
function io_hdf5_has_group

subroutine io_hdf5_initialize
function io_hdf5_is_reference
function io_hdf5_open_dataset
function io_hdf5_open_group

subroutine io_hdf5_read_attribute_-character_1d_array_static
subroutine io_hdf5_read_attribute_-double_1d_array_allocatable
subroutine io_hdf5_read_attribute_-double_scalar
subroutine io_hdf5_read_attribute_-integer8_1d_array_static
subroutine io_hdf5_read_attribute_-integer_1d_array_allocatable
subroutine io_hdf5_read_attribute_-integer_scalar
subroutine io_hdf5_read_attribute_-varstring_1d_array_static
subroutine io_hdf5_read_dataset_-character_1d_array_allocatable
subroutine io_hdf5_read_dataset_-double_1d_array_allocatable
subroutine io_hdf5_read_dataset_-double_2d_array_allocatable
subroutine io_hdf5_read_dataset_-double_3d_array_allocatable
subroutine io_hdf5_read_dataset_-double_4d_array_allocatable
subroutine io_hdf5_read_dataset_-double_5d_array_allocatable
subroutine io_hdf5_read_dataset_-double_6d_array_allocatable
subroutine io_hdf5_read_dataset_-integer8_1d_array_allocatable
subroutine io_hdf5_read_dataset_-integer8_2d_array_allocatable
subroutine io_hdf5_read_dataset_-integer_1d_array_allocatable
subroutine io_hdf5_read_dataset_-integer_2d_array_allocatable
subroutine io_hdf5_read_dataset_-varstring_1d_array_allocatable
subroutine io_hdf5_read_table_-character_1d_array_allocatable

function io_hdf5_is_hdf5
function io_hdf5_open_attribute
subroutine io_hdf5_open_file
subroutine io_hdf5_read_attribute_-character_1d_array_allocatable
subroutine io_hdf5_read_attribute_-character_scalar
subroutine io_hdf5_read_attribute_-double_1d_array_static
subroutine io_hdf5_read_attribute_-integer8_1d_array_allocatable
subroutine io_hdf5_read_attribute_-integer8_scalar
subroutine io_hdf5_read_attribute_-integer_1d_array_static
subroutine io_hdf5_read_attribute_-varstring_1d_array_allocatable
subroutine io_hdf5_read_attribute_-varstring_scalar
subroutine io_hdf5_read_dataset_-character_1d_array_static
subroutine io_hdf5_read_dataset_-double_1d_array_static
subroutine io_hdf5_read_dataset_-double_2d_array_static
subroutine io_hdf5_read_dataset_-double_3d_array_static
subroutine io_hdf5_read_dataset_-double_4d_array_static
subroutine io_hdf5_read_dataset_-double_5d_array_static
subroutine io_hdf5_read_dataset_-double_6d_array_static
subroutine io_hdf5_read_dataset_-integer8_1d_array_static
subroutine io_hdf5_read_dataset_-integer8_2d_array_static
subroutine io_hdf5_read_dataset_-integer_1d_array_static
subroutine io_hdf5_read_dataset_-integer_2d_array_static
subroutine io_hdf5_read_dataset_-varstring_1d_array_static
subroutine io_hdf5_read_table_-integer8_1d_array_allocatable

subroutine io_hdf5_read_table_integer_-1d_array_allocatable
subroutine io_hdf5_read_table_real_1d_-array_allocatable
subroutine io_hdf5_set_defaults
subroutine io_hdf5_uninitialize
subroutine io_hdf5_write_attribute_-character_1d
subroutine io_hdf5_write_attribute_-character_scalar
subroutine io_hdf5_write_attribute_-double_1d
subroutine io_hdf5_write_attribute_-double_2d
subroutine io_hdf5_write_attribute_-double_scalar
subroutine io_hdf5_write_attribute_-integer8_1d
subroutine io_hdf5_write_attribute_-integer8_scalar
subroutine io_hdf5_write_attribute_-integer_1d
subroutine io_hdf5_write_attribute_-integer_scalar
subroutine io_hdf5_write_dataset_-character_1d
subroutine io_hdf5_write_dataset_-double_1d
subroutine io_hdf5_write_dataset_-double_2d
subroutine io_hdf5_write_dataset_-double_3d
subroutine io_hdf5_write_dataset_-double_4d
subroutine io_hdf5_write_dataset_-double_5d
subroutine io_hdf5_write_dataset_-double_6d
subroutine io_hdf5_write_dataset_-integer8_1d
subroutine io_hdf5_write_dataset_-integer8_2d
subroutine io_hdf5_write_dataset_-integer_1d
subroutine io_hdf5_write_dataset_-integer_2d
subroutine io_hdf_assert_is_initialized
subroutine xml_extrapolation_element_-decode

function xml_get_first_element_by_tag_-name
function xml_parse

module mpi_utilities
function counterconstructor
function counterget
function counterincrement
function mpialllogicalscalar
function mpianylogicalscalar
subroutine mpifinalize
subroutine mpiinitialize
function mpisumarrayint
function array_index_double_2d
subroutine array_which
function count_lines_in_file_char
subroutine directory_make_char
subroutine file_unlock
subroutine delete_
function value_
subroutine values_
subroutine hash_perfect_create
function hash_perfect_index
function hash_perfect_is_present
function hash_perfect_size
function hash_perfect_value
function inputparameterget
function inputparameterobjectget
function inputparametersconstructorfilechar
function inputparametersconstructornode

function inputparameterscopiescount
function inputparameterscount
subroutine inputparametersmarkglobal
function inputparametersnode
function inputparameterssubparameters
subroutine inputparametersvalidatename
subroutine inputparametersvaluename
subroutine inputparametersvaluenode

module memory_management
function multicounterconstructor
function multicounterincrement
function multicounterstate
subroutine assert_character_scalar
subroutine assert_double_2d_array
subroutine assert_double_4d_array
subroutine assert_double_complex_1d_-array
subroutine assert_integer8_1d_array
subroutine assert_integer_1d_array
subroutine assert_logical_1d_array
subroutine assert_real_1d_array
subroutine assert_varstring_1d_array
subroutine multicounterappend
function multicountercount
function multicounterisfinal
subroutine assert_character_1d_array
subroutine assert_double_1d_array
subroutine assert_double_3d_array
subroutine assert_double_5d_array
subroutine assert_double_scalar
subroutine assert_integer8_scalar
subroutine assert_integer_scalar
subroutine assert_logical_scalar
subroutine assert_real_scalar
subroutine assert_varstring_scalar

function: galacticus_component_list
Description: Construct a message describing which implementations of a component class provide required functionality.
Code lines: 14
Contained by: module galacticus_error
Modules used: string_handling

subroutine: galacticus_error_handler_register
Description: Register signal handlers.
Code lines: 15
Contained by: module galacticus_error

interface: galacticus_error_report
Code lines: 3
Contained by: module galacticus_error

subroutine: galacticus_error_report_char
Description: Display an error message.
Code lines: 26
Contained by: module galacticus_error

subroutine: galacticus_error_report_varstr
Description: Display an error message.
Code lines: 7
Contained by: module galacticus_error

subroutine: galacticus_error_wait_set
Description: Set the time to wait after an error occurs.
Code lines: 7
Contained by: module galacticus_error

subroutine: galacticus_gsl_error_handler
Description: Handle errors from the GSL library, by flushing all data and then aborting.
Code lines: 52

Contained by: module galacticus_error
Modules used: mpi

subroutine: galacticus_gsl_error_handler_abort_off
Description: Record that we should not abort on GSL errors.
Code lines: 6
Contained by: module galacticus_error

subroutine: galacticus_gsl_error_handler_abort_on
Description: Record that we should abort on GSL errors.
Code lines: 6
Contained by: module galacticus_error

function: galacticus_gsl_error_status
Description: Return current GSL error status.
Code lines: 6
Contained by: module galacticus_error

subroutine: galacticus_signal_handler_sigbus
Description: Handle SIGBUS signals, by flushing all data and then aborting.
Code lines: 43
Contained by: module galacticus_error
Modules used: mpi

subroutine: galacticus_signal_handler_sigfpe
Description: Handle SIGFPE signals, by flushing all data and then aborting.
Code lines: 43
Contained by: module galacticus_error
Modules used: mpi

subroutine: galacticus_signal_handler_sigill
Description: Handle SIGILL signals, by flushing all data and then aborting.
Code lines: 43
Contained by: module galacticus_error
Modules used: mpi

subroutine: galacticus_signal_handler_sigint
Description: Handle SIGINT signals, by flushing all data and then aborting.
Code lines: 43
Contained by: module galacticus_error
Modules used: mpi

subroutine: galacticus_signal_handler_sigsegv
Description: Handle SIGSEGV signals, by flushing all data and then aborting.
Code lines: 43
Contained by: module galacticus_error
Modules used: mpi

subroutine: `galacticus_signal_handler_sigxcpu`
Description: Handle `SIGXCPU` signals, by flushing all data and then aborting.
Code lines: 17
Contained by: module galacticus_error
Modules used: `mpi`

interface: `galacticus_warn`
Code lines: 3
Contained by: module galacticus_error

subroutine: `galacticus_warn_char`
Description: Display a warning message.
Code lines: 30
Contained by: module galacticus_error
Modules used: galacticus_display

subroutine: `galacticus_warn_review`
Description: Review any warning messages emitted during the run.
Code lines: 16
Contained by: module galacticus_error

subroutine: `galacticus_warn_varstr`
Description: Display a warning message
Code lines: 7
Contained by: module galacticus_error

type: `warning`
Code lines: 3
Contained by: module galacticus_error

file: `galacticus.error.wait.F90`
Description: Contains a module which handles setting of error wait times.
Code lines: 50

module: `galacticus_error_wait`
Description: Handle setting of error wait times.
Code lines: 28
Contained by: file galacticus.error.wait.F90
Used by: program galacticus

subroutine: `galacticus_error_wait_set_from_parameters`
Description: Read the parameter that controls the verbosity level, and set that level.
Code lines: 18
Contained by: module galacticus_error_wait
Modules used: galacticus_error input_parameters

file: `galacticus.function_class_destroy.F90`
Description: Contains a module which handles destruction of default `functionClass` objects.

Code lines: 362

module: `galacticus_function_classes_destroys`

Description: Handles resetting of calculations before a new or updated node is processed.
Code lines: 340
Contained by: file galacticus.function_class_destroy.F90
Used by:

program galacticus	subroutine evolveforestsperform
program test_diemerkravtsov2014_-concentration	program test_nfw96_concentration_-dark_energy
program test_prada2011_concentration	program test_zhao2009_flat
program test_zhao2009_dark_energy	program test_zhao2009_open
program test_correa2015_concentration	program test_correa2015_mah

subroutine: `galacticus_function_classes_destroy`

Description: Calls any functions required to destroy all default `functionClass` objects.
Code lines: 330
Contained by: module galacticus_function_classes_destroys
Modules used:

accretion_disk_spectra	accretion_disks
accretion_halo_totals	accretion_halos
atomic_cross_sections_ionization_-photo	atomic_ionization_potentials
atomic_radiation_gaunt_factors	atomic_rates_excitation_collisional
atomic_rates_ionization_collisional	atomic_rates_recombination_-dielectronic
atomic_rates_recombination_radiative	black_hole_binary_initial_separation
black_hole_binary_mergers	black_hole_binary_recoil_velocities
black_hole_binary_separations	chemical_reaction_rates
chemical_states	conditional_mass_functions
cooling_cold_mode_infall_rates	cooling_freefall_times_available
cooling_functions	cooling_infall_radii
cooling_radii	cooling_rates
cooling_specific_angular_momenta	cooling_times
cooling_times_available	cosmological_density_field
cosmology_functions	cosmology_parameters
dark_matter_halo_biases	dark_matter_halo_mass_accretion_-histories
dark_matter_halo_scales	dark_matter_halos_mass_loss_rates
dark_matter_particles	dark_matter_profile_scales
dark_matter_profiles	dark_matter_profiles_concentration
dark_matter_profiles_dmo	dark_matter_profiles_shape
excursion_sets_barriers	excursion_sets_first_crossings
freefall_radii	galactic_dynamics_bar_instabilities
galactic_filters	galactic_structure_solvers
geometry_lightcones	geometry_surveys
gravitational_lensing	halo_mass_functions
halo_model_power_spectrum_modifiers	halo_spin_distributions

hot_halo_cold_mode_density_core_radii
hot_halo_mass_distributions
hot_halo_mass_distributions_core_radii
hot_halo_outflows_reincorporations
hot_halo_ram_pressure_forces
hot_halo_ram_pressure_stripping
hot_halo_ram_pressure_stripping_timescales
hot_halo_temperature_profiles
intergalactic_medium_filtering_masses
intergalactic_medium_state
linear_growth
mass_distributions
mass_function_incompletenesses
math_operators_unary
merger_tree_branching
merger_tree_branching_modifiers
merger_tree_construction
merger_tree_operators
merger_tree_outputters
merger_tree_read_importers
merger_tree_timesteps
merger_tree_walkers
merger_trees_build_mass_resolution
merger_trees_build_masses
merger_trees_build_masses_distributions
merger_trees_builders
merger_trees_evolve
merger_trees_evolve_node
merger_trees_merge_node
meta_tree_compute_times
model_parameters
models_likelihoods
nbody_importers
nbody_operators
node_property_extractors
output_analyses
output_analysis_distribution_normalizers
output_analysis_distribution_operators
output_analysis_molecular_ratios
output_analysis_property_operators
output_analysis_weight_operators
output_times
posterior_sample_differential_proposal_size
posterior_sample_differential_random_jump
posterior_sampling_convergence
posterior_sampling_prop_size_temp_exp
posterior_sampling_simulation
posterior_sampling_state
posterior_sampling_state_initialize
posterior_sampling_stopping_criteria
power_spectra
power_spectra_nonlinear
power_spectra_primordial
power_spectra_primordial_transferred
power_spectrum_window_functions
radiation_fields
ram_pressure_stripping_mass_loss_rate_disks
ram_pressure_stripping_mass_loss_rate_spheroids
satellite_dynamical_friction
satellite_merging_mass_movements
satellite_merging_progenitor_properties
satellite_merging_remnant_sizes
satellite_merging_timescales
satellite_oprhan_distributions
satellite_tidal_heating
satellite_tidal_stripping
satellites_tidal_fields
star_formation_feedback_disks
star_formation_feedback_expulsion_disks
star_formation_feedback_expulsion_spheroids
star_formation_feedback_spheroids
star_formation_histories
star_formation_rate_surface_density_disks
star_formation_timescales_disks

star_formation_timescales_spheroids
statistics_nbody_halo_mass_errors
stellar_astrophysics
stellar_astrophysics_tracks
stellar_astrophysics_winds
stellar_feedback
stellar_population_properties
stellar_population_selectors
stellar_population_spectra
stellar_population_spectra_postprocess
stellar_populations
stellar_populations_initial_mass_functions
stellar_spectra_dust_attenuations
supernovae_population_iii
supernovae_type_ia
task_evolve_forests_work_shares
tidal_stripping_mass_loss_rate_disks
tidal_stripping_mass_loss_rate_spheroids
transfer_functions
unevolved_subhalo_mass_functions
universe_operators
virial_density_contrast
virial_orbits

file: galacticus.input.path.F90
Description: Contains a module which provides the path for Galacticus inputs and scripts.
Code lines: 95

module: galacticus_paths
Description: Provides the path for Galacticus inputs and scripts.
Code lines: 73
Contained by: file galacticus.input.path.F90
Modules used: iso_varying_string
Used by:

subroutine hopkins2007buildfile	function hopkins2007constructorinternal
subroutine atomic_data_initialize	program benchmark_stellar_populations_luminosities
function ciefileconstructorparameters	subroutine atomiccliecloudytabulate
subroutine atomiccliecloudytabulate	function blackholebulgerelationconstructorinternal
function colordistributionsdssconstructorinternal	function concentrationdistributioncdmcococonstructorinter
function concentrationvshalomasscdmludlow2016constructorinternal	function correlationfunctionhearin2013sdssconstructorinte
function disksizeinclinationconstructorinternal	function galaxysizessdssconstructorinternal
function luminosityfunctiongunawardhana2013sdssconstructorinternal	function luminosityfunctionsobral2013hizelsconstructorint
function luminosityfunctionmonterodorta2009sdssconstructorinternal	function massfunctionalfamartin2010constructorintern
function massfunctionstellarbernardi2013sdssconstructorinternal	function massfunctionstellarbaldry2012gamaconstructorinte
function massfunctionstellarprimusconstructorinternal	function massfunctionstellarsdssconstructorinternal
function massfunctionstellarukidssudsconstructorinternal	function massfunctionstellarultravistaconstructorinternal

function massfunctionstellarvipersconstructorinternal
function massfunctionstellarzfourgeconstructorinternal
function massmetallicityandrews2013constructorinternal
function massmetallicityblanc2017constructorinternal
function morphologicalfractiongamamoffett2016constructorinternal
function quenchedfractionbnbett2007constructorinternal
function stellarvshalomassrelationleauthaud2012constructorinternal
subroutine galacticus_build_output
function geometrymangleangularpower
subroutine geometrymanglebuild
function geometrymanglesolidangle
file geometry.surveys.Baldry-2012-GAMA.F90
file geometry.surveys.Bernardi-2013-SDSS.F90
subroutine caputi2011ukidssudsrandomsinitialize
file geometry.surveys.Davidzon-2013-VIPERS.F90
file geometry.surveys.Gunawardhana-2013-SDSS.F90
file geometry.surveys.Kelvin-2014-GAMAnear.F90
subroutine liwhite2009sdssrandomsinitialize
file geometry.surveys.Local_Group_-DES.F90
file geometry.surveys.Local_Group_-classical.F90
subroutine martin2010alfalfarandomsinitialize
file geometry.surveys.Montero-Dorta-2009-SDSS.F90
file geometry.surveys.Moustakas-2013-PRIMUS.F90
file geometry.surveys.Muzzin-2013-ULTRAVISTA.F90
file geometry.surveys.Tomczak-2014-ZFOURGE.F90
file geometry.surveys.mangle.F90
subroutine emissionlinedatabaseinitialize
subroutine filter_response_load
subroutine interface_camb_initialize
subroutine interface_camb_transfer_-function
subroutine interface_cloudy_initialize
subroutine interface_fsps_initialize
function localgroupdbconstructorinternal
subroutine localgroupdbupdate
subroutine interface_recfast_initialize
function recfastconstructorinternal
function massfunctionconstructorinternal
function projectedcorrelationfunctionconstructorinternal
function lmnstyemssnlineconstructorinternal
subroutine chemical_structure_-initialize
function intergalacticbackgroundfileconstructorinternal
function fileconstructorparameters
function hegerwoosley2002constructorinternal
function nagashima2005constructorinternal
function fileconstructorparameters
subroutine stellar_population_-luminosity_tabulate
function fspsconstructorinternal
function fileconstructorparameters
function standardinterpolate
function barkana2001wdmconstructorinternal

function farahiconstructorinternal
subroutine takahashi2011lensingdistributionconstruct
function rodriguezpuebla2016constructorinternal
function tinker2008constructorinternal
function cosmicemuvalue
subroutine spherical_collape_matter_lambda_delta_virial_tabulate
subroutine spherical_collapse_matter_lambda_critical_overdensity_tabulate
program test_diemerkravtsov2014_concentration
program test_zhao2009_flat
program test_zhao2009_dark_energy
program test_zhao2009_open
program test_correa2015_concentration
program test_cooling_functions
program test_correa2015_mah
program test_stellar_populations
program test_stellar_populations_luminosities
function file_name_expand
function inputparametersconstructornode

function: galacticuspath
Description: Returns the path to various Galacticus resources.
Code lines: 34
Contained by: module galacticus_paths

subroutine: pathsretrieve
Description: Retrieve the Galacticus input data path from the environment.
Code lines: 14
Contained by: module galacticus_paths

file: galacticus.meta.evolver_profiler.F90
Description: Contains a module which constructs a profile of Galacticus ODE evolver statistics.
Code lines: 152

module: galacticus_meta_evolver_profiler
Description: Constructs a profile of Galacticus ODE evolver statistics.
Code lines: 130
Contained by: file galacticus.meta.evolver_profiler.F90
Modules used: hashes
Used by: subroutine galacticus_output_close_file, subroutine standardsteperroranalyzer

subroutine: galacticus_meta_evolver_profile
Description: Record profiling information on the ODE evolver.
Code lines: 66
Contained by: module galacticus_meta_evolver_profiler
Modules used: arrays_search, input_parameters, iso_c_binding, iso_varying_string, memory_management, numerical_ranges

subroutine: galacticus_meta_evolver_profiler_output
Description: Outputs collected meta-data on tree evolution.
Code lines: 38
Contained by: module galacticus_meta_evolver_profiler

Modules used: galacticus_hdf5 iso_varying_string numerical_constants_astronomical

file: galacticus.meta.tree_timing.F90
Description: Contains a module which records and outputs timing and memory usage data for processing trees.
Code lines: 298

module: galacticus_meta_tree_timing
Description: Records and outputs timing data for processing trees.
Code lines: 276
Contained by: file galacticus.meta.tree_timing.F90
Modules used: iso_c_binding kind_numbers
Used by: subroutine galacticus_output_close_file subroutine standardevolve subroutine evolveforestsperform

subroutine: meta_tree_timing_initialize
Description: Initialize the tree timing meta-data module.
Code lines: 34
Contained by: module galacticus_meta_tree_timing
Modules used: galacticus_error input_parameters

subroutine: meta_tree_timing_output
Description: Outputs collected meta-data on tree evolution.
Code lines: 33
Contained by: module galacticus_meta_tree_timing
Modules used: galacticus_hdf5 numerical_constants_astronomical

subroutine: meta_tree_timing_post_evolve
Description: Record memory usage.
Code lines: 26
Contained by: module galacticus_meta_tree_timing
Modules used: galacticus_nodes merger_tree_walkers

subroutine: meta_tree_timing_post_tree_evolve
Description: Record the CPU time after evolving a tree.
Code lines: 70
Contained by: module galacticus_meta_tree_timing
Modules used: memory_management

subroutine: meta_tree_timing_pre_construction
Description: Record the CPU time prior to construction of a tree.
Code lines: 21
Contained by: module galacticus_meta_tree_timing

subroutine: meta_tree_timing_pre_evolve
Description: Record the CPU time prior to evolving tree.
Code lines: 35
Contained by: module galacticus_meta_tree_timing

Modules used: galacticus_nodes merger_tree_walkers

file: galacticus.output.HDF5.F90
Description: Contains a module which manages HDF5 output from Galacticus.
Code lines: 50

module: galacticus_hdf5
Description: Manages HDF5 output from Galacticus.
Code lines: 28
Contained by: file galacticus.output.HDF5.F90
Modules used: hdf5 io_hdf5
Used by: program galacticus
subroutine galacticus_meta_evolver_profiler_output
subroutine meta_tree_timing_output
module galacticus_output_open
subroutine localgroupmassfunctionfinalize
subroutine correlationfunctionfinalize
subroutine meanfunction1dfinalize
subroutine morphologicalfractiongamamoffett2016finalize
subroutine scatterfunction1dfinalize
subroutine volumefunction1dfinalize
subroutine galacticus_build_output
subroutine galacticus_extra_output_halo_fourier_profile
subroutine galacticus_output_halo_model_initialize
subroutine galacticus_version_output
subroutine filter_response_load
subroutine historywrite
subroutine recordevolutionoutput
subroutine augmentfinalize
subroutine conditionalmffinalize
subroutine informationcontentfinalize
subroutine massaccretionhistoryoperate
subroutine outputrootmassesfinalize
subroutine profilerfinalize
subroutine merger_tree_structure_output
subroutine analyzerfinalize
subroutine standardoutputgroupcreate
subroutine node_component_black_hole_standard_output_merger
subroutine node_component_black_hole_standard_output_properties
subroutine node_component_dynamics_statistics_bars_output
subroutine node_component_merging_statistics_major_output
function intergalacticbackgroundinternalupdate
subroutine satellite_merging_output
subroutine insituoutput
subroutine metallicitysplitoutput
subroutine catalogprojectedcorrelationfunctionperform
subroutine conditionalmassfunctionperform
function excursionsetsconstructorparameters
subroutine excursionsetsperform
subroutine halomassfunctionperform
subroutine halomodelprojectedcorrelationfunctionperform
subroutine halospindistributionperform
subroutine intergalacticmediumstateperform
subroutine powerspectraperform
function intergalacticmediumstateevolveupdate

subroutine openmp_critical_wait_times

file: galacticus.output.HDF5.open.F90
Description: Contains a module which handles opening of the GALACTICUS output file.
Code lines: 246

module: galacticus_output_open
Description: Handles opening of the GALACTICUS output file.
Code lines: 224
Contained by: file galacticus.output.HDF5.open.F90
Modules used: galacticus_hdf5, iso_varying_string
Used by: program galacticus, program test_stellar_populations_luminosities

subroutine: galacticus_output_close_file
Description: Close the GALACTICUS output file.
Code lines: 57
Contained by: module galacticus_output_open
Modules used: events_hooks, galacticus_meta_evolver_profiler, galacticus_meta_tree_timing, merger_tree_output_structure, merger_trees_dump_evolution, openmp_utilities, system_command

subroutine: galacticus_output_open_file
Description: Open the file for GALACTICUS output.
Code lines: 150
Contained by: module galacticus_output_open
Modules used: galacticus_build, galacticus_versioning, hdf5, input_parameters, iso_c_binding, mpi_utilities, string_handling

file: galacticus.output.analyses.F90
Description: Contains a module which provides a class that implements on-the-fly analyses.
Code lines: 61

module: output_analyses
Description: Provides a class that implements on-the-fly analyses.
Code lines: 39
Contained by: file galacticus.output.analyses.F90
Modules used: galacticus_nodes, iso_c_binding
Used by: subroutine galacticus_function_classes_destroy, subroutine galacticus_state_retrieve, subroutine galacticus_state_store, file merger_trees.outputter.analyzer.F90, file models.likelihoods.galaxy_population.F90

file: galacticus.output.analyses.HI_vs_halo_mass_relation.ALFALFA_Padmanabhan_2017.F90
Description: Contains a module which implements an HI vs halo mass relation analysis class.
Code lines: 333
Modules used: iso_c_binding

function: hivshalomassrelationpadmanabhan2017constructorinternal
Description: Constructor for the "hiVsHaloMassRelationPadmanabhan2017" output analysis class for internal use.
Code lines: 237
Contained by: file galacticus.output.analyses.HI_vs_halo_mass_relation.ALFALFA_Padmanabhan_2017.F90
Modules used: cosmology_functions, cosmology_parameters, dark_matter_halo_scales, galacticus_error, galacticus_nodes, iso_varying_string, memory_management, node_property_extractors, numerical_comparison, numerical_constants_astronomical, numerical_ranges, output_analysis_distribution_operators, output_analysis_molecular_ratios, output_analysis_property_operators, output_analysis_utilities, output_analysis_weight_operators, output_times, string_handling, virial_density_contrast

function: hivshalomassrelationpadmanabhan2017constructorparameters
Description: Constructor for the "hiVsHaloMassRelationPadmanabhan2017" output analysis class which takes a parameter set as input.
Code lines: 53
Contained by: file galacticus.output.analyses.HI_vs_halo_mass_relation.ALFALFA_Padmanabhan_2017.F90
Modules used: cosmology_functions, cosmology_parameters, functions_global, input_parameters, output_analysis_molecular_ratios

interface: outputanalysishivshalomassrelationpadmanabhan2017
Description: Constructors for the "hiVsHaloMassRelationPadmanabhan2017" output analysis class.
Code lines: 4
Contained by: file galacticus.output.analyses.HI_vs_halo_mass_relation.ALFALFA_Padmanabhan_2017.F90

file: galacticus.output.analyses.Local_Group_mass_functions.F90
Description: Contains a module which implements an output analysis class that computes mass functions for Local Group satellite galaxies.
Code lines: 562

subroutine: localgroupmassfunctionanalyze
Description: Implement a localGroupMassFunction output analysis.
Code lines: 11
Contained by: file galacticus.output.analyses.Local_Group_mass_functions.F90

function: localgroupmassfunctionconstructorinternal
Description: Constructor for the "localGroupMassFunction" output analysis class for internal use.

Code lines: 266
Contained by: file galacticus.output.analyses.Local_Group_mass_functions.F90
Modules used: galactic_filters, geometry_surveys, interface_local_group_db, node_property_extractors, numerical_comparison, numerical_constants_astronomical, numerical_ranges, output_analysis_distribution_-normalizers, output_analysis_distribution_-operators, output_analysis_property_operators, output_analysis_weight_operators, output_times

function: localgroupmassfunctionconstructorparameters
Description: Constructor for the "localGroupMassFunction" output analysis class which takes a parameter set as input.
Code lines: 100
Contained by: file galacticus.output.analyses.Local_Group_mass_functions.F90
Modules used: input_parameters, output_times

subroutine: localgroupmassfunctiondestructor
Description: Destructor for the localGroupMassFunction output analysis class.
Code lines: 8
Contained by: file galacticus.output.analyses.Local_Group_mass_functions.F90

subroutine: localgroupmassfunctionfinalize
Description: Implement a localGroupMassFunction output analysis finalization.
Code lines: 37
Contained by: file galacticus.output.analyses.Local_Group_mass_functions.F90
Modules used: galacticus_hdf5, io_hdf5, numerical_constants_astronomical

subroutine: localgroupmassfunctionfinalizeanalysis
Description: Finalize analysis of a localGroupMassFunction output analysis.
Code lines: 26
Contained by: file galacticus.output.analyses.Local_Group_mass_functions.F90

function: localgroupmassfunctionloglikelihood
Description: Return the log-likelihood of a localGroupMassFunction output analysis. The likelihood function assumes that the model prediction for the number of satellite galaxies in any given mass bin follows a negative binomial distribution as was found for dark matter subhalos [Boylan-Kolchin et al., 2010, see also Lu et al. 2016]. This has been confirmed by examining the results of many tree realizations, although it in principal could be model-dependent.
Code lines: 21
Contained by: file galacticus.output.analyses.Local_Group_mass_functions.F90
Modules used: statistics_distributions_discrete

subroutine: localgroupmassfunctionreduce
Description: Implement a localGroupMassFunction output analysis reduction.
Code lines: 15
Contained by: file galacticus.output.analyses.Local_Group_mass_functions.F90

Modules used: galacticus_error

interface: `outputanalysislocalgroupmassfunction`
Description: Constructors for the "localGroupMassFunction" output analysis class.
Code lines: 4
Contained by: file galacticus.output.analyses.Local_Group_mass_functions.F90

file: `galacticus.output.analyses.black_hole_bulge_relation.F90`
Description: Contains a module which implements a mass-metallicity relation analysis class.
Code lines: 262

function: `blackholebulgerelationconstructorinternal`
Description: Constructor for the "blackHoleBulgeRelation" output analysis class for internal use.
Code lines: 161
Contained by: file galacticus.output.analyses.black_hole_bulge_relation.F90
Modules used: cosmology_functions, cosmology_parameters, galacticus_error, galacticus_paths, io_hdf5, memory_management, node_property_extractors, numerical_comparison, numerical_constants_astronomical, numerical_ranges, output_analysis_distribution_operators, output_analysis_property_operators, output_analysis_utilities, output_analysis_weight_operators, output_times

function: `blackholebulgerelationconstructorparameters`
Description: Constructor for the "blackHoleBulgeRelation" output analysis class which takes a parameter set as input.
Code lines: 60
Contained by: file galacticus.output.analyses.black_hole_bulge_relation.F90
Modules used: cosmology_functions, input_parameters

interface: `outputanalysisblackholebulgerelation`
Description: Constructors for the "blackHoleBulgeRelation" output analysis class.
Code lines: 4
Contained by: file galacticus.output.analyses.black_hole_bulge_relation.F90

file: `galacticus.output.analyses.color_distribution.SDSS.F90`
Description: Contains a module which implements a color distribution output analysis class for SDSS data.
Code lines: 255
Modules used: cosmology_functions

function: `colordistributionsdssconstructorinternal`
Description: Internal constructor for the "colorDistributionSDSS" output analysis class.
Code lines: 176
Contained by: file galacticus.output.analyses.color_distribution.SDSS.F90
Modules used: cosmology_parameters, galacticus_error, galacticus_paths, io_hdf5

iso_varying_string
numerical_comparison
numerical_constants_prefixes
output_analysis_utilities
memory_management
numerical_constants_astronomical
output_analyses_options
output_times

function: colordistributionsdssconstructorparameters
Description: Constructor for the "colorDistributionSDSS" output analysis class which takes a parameter set as input.
Code lines: 24
Contained by: file galacticus.output.analyses.color_distribution.SDSS.F90
Modules used: input_parameters

subroutine: colordistributionsdssdestructor
Description: Destructor for the "colorDistributionSDSS" output analysis class.
Code lines: 7
Contained by: file galacticus.output.analyses.color_distribution.SDSS.F90

interface: outputanalysiscolordistributionsdss
Description: Constructors for the "colorDistributionSDSS" output analysis class.
Code lines: 4
Contained by: file galacticus.output.analyses.color_distribution.SDSS.F90

file: galacticus.output.analyses.concentration_distribution.CDM.COCO.F90
Description: Contains a module which implements a concentration distribution output analysis class for COCO CDM data.
Code lines: 301

function: concentrationdistributioncdmcococonstructorinternal
Description: Internal constructor for the "concentrationDistributionCDMCOCO" output analysis class.
Code lines: 218
Contained by: file galacticus.output.analyses.concentration_distribution.CDM.COCO.F90
Modules used: cosmology_functions, cosmology_parameters, galacticus_error, galacticus_paths, io_hdf5, iso_varying_string, memory_management, numerical_comparison, output_analyses_options, output_analysis_utilities, output_times, statistics_nbody_halo_mass_errors, virial_density_contrast

function: concentrationdistributioncdmcococonstructorparameters
Description: Constructor for the "concentrationDistributionCDMCOCO" output analysis class which takes a parameter set as input.
Code lines: 41
Contained by: file galacticus.output.analyses.concentration_distribution.CDM.COCO.F90
Modules used: cosmology_functions, cosmology_parameters, input_parameters, statistics_nbody_halo_mass_errors

interface: outputanalysisconcentrationdistributioncdmcoco
Description: Constructors for the "concentrationDistributionCDMCOCO" output analysis class.
Code lines: 4

Contained by: file galacticus.output.analyses.concentration_distribution.CDM.COCO.F90

file: galacticus.output.analyses.concentration_vs_mass_relation.CDM.Ludlow_2016.F90
Description: Contains a module which implements a concentration vs. halo mass analysis class matched to the Ludlow et al. [2016] CDM sample.
Code lines: 180
Modules used: statistics_nbody_halo_mass_errors

function: concentrationvshalomasscdmludlow2016constructorinternal
Description: Constructor for the "concentrationVsHaloMassCDMLudlow2016" output analysis class for internal use.
Code lines: 109
Contained by: file galacticus.output.analyses.concentration_vs_mass_relation.CDM.Ludlow_2016.F90
Modules used:
cosmology_functions
cosmology_parameters
galacticus_error
galacticus_paths
io_hdf5
iso_varying_string
memory_management
node_property_extractors
numerical_comparison
numerical_constants_astronomical
output_analysis_distribution_operators
output_analysis_property_operators
output_analysis_utilities
output_analysis_weight_operators
output_times
virial_density_contrast

function: concentrationvshalomasscdmludlow2016constructorparameters
Description: Constructor for the "concentrationVsHaloMassCDMLudlow2016" output analysis class which takes a parameter set as input.
Code lines: 28
Contained by: file galacticus.output.analyses.concentration_vs_mass_relation.CDM.Ludlow_2016.F90
Modules used:
cosmology_functions
cosmology_parameters
input_parameters

interface: outputanalysisconcentrationvshalomasscdmludlow2016
Description: Constructors for the "concentrationVsHaloMassCDMLudlow2016" output analysis class.
Code lines: 4
Contained by: file galacticus.output.analyses.concentration_vs_mass_relation.CDM.Ludlow_2016.F90

file: galacticus.output.analyses.correlation_function.F90
Description: Contains a module which implements a generic two-point correlation function output analysis class.
Code lines: 1144
Modules used:
cosmology_functions
dark_matter_halo_biases
dark_matter_profiles_dmo
galactic_filters
geometry_surveys
halo_model_power_spectrum_modifiers
iso_c_binding
linear_growth
node_property_extractors
output_analysis_distribution_operators
output_analysis_property_operators
output_times
power_spectra

subroutine: `correlationfunctionaccumulatehalo`

Description: Assumulate a single halo's contributions to the halo model one- and two-halo terms. For the one-halo term we count contributions from central-satellite pairs, and from satellite-satellite pairs. Contributions differ in the scalings applied to the Fourier-transformed dark matter halo density profile—see [Cooray and Sheth, 2002, §6.1] for a discussion of this. The number of satellites in the halo is assumed to follow a Poisson binomial distribution.

Code lines: 160

Contained by: file `galacticus.output.analyses.correlation_function.F90`

Modules used: `galacticus_nodes` `halo_model_power_spectrum_modifiers` `linear_algebra` `math_distributions_poisson_binomial` `vectors`

subroutine: `correlationfunctionaccumulatenode`

Description: Accumulate a single galaxy to the population of the current halo. Since galaxy masses have random errors, each galaxy added is assigned an inclusion probability, which will be taken into account when evaluating the one- and two-halo terms from this halo in the halo model.

Code lines: 42

Contained by: file `galacticus.output.analyses.correlation_function.F90`

subroutine: `correlationfunctionanalyze`

Description: Implement a correlationFunction output analysis.

Code lines: 28

Contained by: file `galacticus.output.analyses.correlation_function.F90`

function: `correlationfunctionconstructorfile`

Description: Constructor for the "correlation" output analysis class which reads bin information from a standard format file.

Code lines: 56

Contained by: file `galacticus.output.analyses.correlation_function.F90`

Modules used: `cosmology_functions` `cosmology_parameters` `io_hdf5`

function: `correlationfunctionconstructorinternal`

Description: Constructor for the "correlationFunction" output analysis class for internal use.

Code lines: 83

Contained by: file `galacticus.output.analyses.correlation_function.F90`

Modules used: `galacticus_error` `iso_c_binding` `memory_management` `numerical_ranges` `output_analysis_utilities`

function: `correlationfunctionconstructorparameters`

Description: Constructor for the "correlationFunction" output analysis class which takes a parameter set as input.

Code lines: 246

Contained by: file `galacticus.output.analyses.correlation_function.F90`

Modules used: `galacticus_error` `input_parameters`

subroutine: `correlationfunctiondestructor`

Description: Destructor for the "correlationFunction" output analysis class.

Code lines: 19
Contained by: file galacticus.output.analyses.correlation_function.F90

subroutine: correlationfunctionfinalize
Description: Implement a correlationFunction output analysis finalization.
Code lines: 57
Contained by: file galacticus.output.analyses.correlation_function.F90
Modules used: galacticus_hdf5 io_hdf5 numerical_constants_astronomical

subroutine: correlationfunctionfinalizeanalysis
Description: Compute final covariances and normalize.
Code lines: 264
Contained by: file galacticus.output.analyses.correlation_function.F90
Modules used: fftlogs linear_algebra memory_management mpi_utilities numerical_constants_math table_labels tables vectors

function: binningintegrandweight
Description: The weight function applied to the projected correlation function when integrating into bins.
Code lines: 7
Contained by: subroutine correlationfunctionfinalizeanalysis

function: projectionintegrandweight
Description: The weight function applied to the correlation function when integrating to get the projected correlation function.
Code lines: 11
Contained by: subroutine correlationfunctionfinalizeanalysis

function: correlationfunctionloglikelihood
Description: Return the log-likelihood of a correlationFunction output analysis.
Code lines: 31
Contained by: file galacticus.output.analyses.correlation_function.F90
Modules used: galacticus_error linear_algebra numerical_constants_math

subroutine: correlationfunctionreduce
Description: Implement a correlationFunction output analysis reduction.
Code lines: 23
Contained by: file galacticus.output.analyses.correlation_function.F90
Modules used: galacticus_error

subroutine: correlationfunctiontermindices
Description: Return the indices in the term covariances array at which one-halo, two-halo, and density terms are stored for the given mass.
Code lines: 11
Contained by: file galacticus.output.analyses.correlation_function.F90

interface: outputanalysiscorrelationfunction

Description: Constructors for the "correlationFunction" output analysis class.
Code lines: 5
Contained by: file galacticus.output.analyses.correlation_function.F90

file: galacticus.output.analyses.correlation_function.Hearin2014_SDSS.F90
Description: Contains a module which implements a correlation function output analysis class for the Hearin et al. [2013] analysis.
Code lines: 261

function: correlationfunctionhearin2013sdssconstructorinternal
Description: Constructor for the "correlationFunctionHearin2013SDSS" output analysis class for internal use.
Code lines: 112
Contained by: file galacticus.output.analyses.correlation_function.Hearin2014_SDSS.F90
Modules used: cosmology_parameters, galacticus_paths, iso_c_binding, output_analysis_property_operators

function: correlationfunctionhearin2013sdssconstructorparameters
Description: Constructor for the "correlationFunctionHearin2013SDSS" output analysis class which takes a parameter set as input.
Code lines: 109
Contained by: file galacticus.output.analyses.correlation_function.Hearin2014_SDSS.F90
Modules used: input_parameters, iso_c_binding

interface: outputanalysiscorrelationfunctionhearin2013sdss
Description: Constructors for the "correlationFunctionHearin2013SDSS" output analysis class.
Code lines: 4
Contained by: file galacticus.output.analyses.correlation_function.Hearin2014_SDSS.F90

file: galacticus.output.analyses.distribution_normalizer.F90
Description: Contains a module which provides a class that normalizors on distributions used in on-the-fly output analyses.
Code lines: 42

module: output_analysis_distribution_normalizers
Description: Provides a class that normalizors on distributions used in on-the-fly output analyses.
Code lines: 20
Contained by: file galacticus.output.analyses.distribution_normalizer.F90
Modules used: galacticus_nodes
Used by: subroutine galacticus_function_classes_destroy, function localgroupmassfunctionconstructorinternal, file galacticus.output.analyses.mean_function_1d.F90, file galacticus.output.analyses.scatter_function_1d.F90, file galacticus.output.analyses.volume_function_1d.F90, subroutine galacticus_state_retrieve

subroutine galacticus_state_store

file: galacticus.output.analyses.distribution_normalizer.bin_width.F90
Description: Contains a module which implements a bin width output analysis distribution normalizer class.
Code lines: 67

function: binwidthconstructorparameters
Description: Constructor for the "binWidth" output analysis distribution normalizer class which takes a parameter set as input.
Code lines: 10
Contained by: file galacticus.output.analyses.distribution_normalizer.bin_width.F90
Modules used: input_parameters

subroutine: binwidthnormalize
Description: Implement a bin width output analysis distribution normalizer.
Code lines: 16
Contained by: file galacticus.output.analyses.distribution_normalizer.bin_width.F90
Modules used: iso_c_binding

interface: outputanalysisdistributionnormalizerbinwidth
Description: Constructors for the "binWidth" output analysis distribution normalizer class.
Code lines: 3
Contained by: file galacticus.output.analyses.distribution_normalizer.bin_width.F90

file: galacticus.output.analyses.distribution_normalizer.identity.F90
Description: Contains a module which implements an identity output analysis distribution normalizer class.
Code lines: 62

function: identityconstructorparameters
Description: Constructor for the "identity" output analysis distribution normalizer class which takes a parameter set as input.
Code lines: 10
Contained by: file galacticus.output.analyses.distribution_normalizer.identity.F90
Modules used: input_parameters

subroutine: identitynormalize
Description: Implement a bin width output analysis distribution normalizer.
Code lines: 11
Contained by: file galacticus.output.analyses.distribution_normalizer.identity.F90

interface: outputanalysisdistributionnormalizeridentity
Description: Constructors for the "identity" output analysis distribution normalizer class.
Code lines: 3
Contained by: file galacticus.output.analyses.distribution_normalizer.identity.F90

file: galacticus.output.analyses.distribution_normalizer.log10_to_log.F90
Description: Contains a module which implements a $\log_{10} \rightarrow$ log output analysis distribution normalizer class.
Code lines: 63

function: `log10tologconstructorparameters`
Description: Constructor for the "log10ToLog" output analysis distribution normalizer class which takes a parameter set as input.
Code lines: 10
Contained by: file `galacticus.output.analyses.distribution_normalizer.log10_to_log.F90`
Modules used: `input_parameters`

subroutine: `log10tolognormalize`
Description: Implement a bin width output analysis distribution normalizer.
Code lines: 12
Contained by: file `galacticus.output.analyses.distribution_normalizer.log10_to_log.F90`

interface: `outputanalysisdistributionnormalizerlog10tolog`
Description: Constructors for the "log10ToLog" output analysis distribution normalizer class.
Code lines: 3
Contained by: file `galacticus.output.analyses.distribution_normalizer.log10_to_log.F90`

file: `galacticus.output.analyses.distribution_normalizer.sequence.F90`
Description: Contains a module which implements a sequence of normalizers on on-the-fly outputs.
Code lines: 155

type: `normalizerlist`
Code lines: 3
Contained by: file `galacticus.output.analyses.distribution_normalizer.sequence.F90`

interface: `outputanalysisdistributionnormalizersequence`
Description: Constructors for the sequence on-the-fly output distribution normalizer class.
Code lines: 4
Contained by: file `galacticus.output.analyses.distribution_normalizer.sequence.F90`

function: `sequenceconstructorinternal`
Description: Internal constructor for the sequence merger tree normalizer class.
Code lines: 14
Contained by: file `galacticus.output.analyses.distribution_normalizer.sequence.F90`

function: `sequenceconstructorparameters`
Description: Constructor for the sequence on-the-fly output normalizer class which takes a parameter set as input.
Code lines: 22
Contained by: file `galacticus.output.analyses.distribution_normalizer.sequence.F90`
Modules used: `input_parameters`

subroutine: `sequencedeepcopy`
Description: Perform a deep copy for the `sequence` output analysis distribution normalizer operator class.
Code lines: 31
Contained by: file `galacticus.output.analyses.distribution_normalizer.sequence.F90`
Modules used: `galacticus_error`

subroutine: `sequencedestructor`
Description: Destructor for the merger tree normalizer function class.
Code lines: 16
Contained by: file galacticus.output.analyses.distribution_normalizer.sequence.F90

subroutine: `sequencenormalize`
Description: Perform a sequence normalization on an on-the-fly output distribution.
Code lines: 15
Contained by: file galacticus.output.analyses.distribution_normalizer.sequence.F90

file: `galacticus.output.analyses.distribution_normalizer.unitarity.F90`
Description: Contains a module which implements a unitarity output analysis distribution normalizer class.
Code lines: 68

interface: `outputanalysisdistributionnormalizerunitarity`
Description: Constructors for the "unitarity" output analysis distribution normalizer class.
Code lines: 3
Contained by: file galacticus.output.analyses.distribution_normalizer.unitarity.F90

function: `unitarityconstructorparameters`
Description: Constructor for the "unitarity" output analysis distribution normalizer class which takes a parameter set as input.
Code lines: 10
Contained by: file galacticus.output.analyses.distribution_normalizer.unitarity.F90
Modules used: input_parameters

subroutine: `unitaritynormalize`
Description: Implement a unitarity output analysis distribution normalizer.
Code lines: 17
Contained by: file galacticus.output.analyses.distribution_normalizer.unitarity.F90

file: `galacticus.output.analyses.distribution_operator.F90`
Description: Contains a module which provides a class that operators on distributions used in on-the-fly output analyses.
Code lines: 55

module: `output_analysis_distribution_operators`
Description: Provides a class that operators on distributions used in on-the-fly output analyses.
Code lines: 33
Contained by: file galacticus.output.analyses.distribution_operator.F90
Modules used: galacticus_nodes, iso_c_binding
Used by: subroutine galacticus_function_classes_destroy, function hivshalomassrelationpadmanabhan2017constructorin, function localgroupmassfunctionconstructorinterna, function blackholebulgerelationconstructorinternal, function concentrationvshalomasscdmludlow2016const, file galacticus.output.analyses.correlation_function.F90

function luminosityfunctiongunawardhana2013sdssconstructorinternal
function luminosityfunctionmonterodorta2009sdssconstructorinternal
function massfunctionstellarbernardi2013sdssconstructorinternal
function massfunctionstellarprimusconstructorinternal
function massfunctionstellarukidssudsconstructorinternal
function massfunctionstellarvipersconstructorinternal
function massmetallicityandrews2013constructorinternal
file galacticus.output.analyses.mean_function_1d.F90
file galacticus.output.analyses.scatter_function_1d.F90
file galacticus.output.analyses.volume_function_1d.F90
subroutine galacticus_state_store
function luminosityfunctionsobral2013hizelsconstructorinternal
function massfunctionstellaralfamartin2010constructorinternal
function massfunctionstellarbaldry2012gamaconstructorinternal
function massfunctionstellarsdssconstructorinternal
function massfunctionstellarultravistaconstructorinternal
function massfunctionstellarzfourgeconstructorinternal
function massmetallicityblanc2017constructorinternal
function morphologicalfractiongamamoffett2016constructorinternal
function stellarvshalomassrelationleauthaud2012constructorinternal
subroutine galacticus_state_retrieve

file: galacticus.output.analyses.distribution_operator.disk_size_inclination.F90

Description: Contains a module which implements the effects of inclination on disk size in an output analysis distribution operator class.
Code lines: 206
Modules used: tables

function: disksizeinclinationconstructorinternal

Description: Internal constructor for the "diskSizeInclination" output analysis distribution operator class.
Code lines: 61
Contained by: file galacticus.output.analyses.distribution_operator.disk_size_inclination.F90
Modules used: file_utilities, galacticus_paths, io_hdf5, iso_varying_string, numerical_constants_math, root_finder, table_labels

function: disksizeinclinationconstructorparameters

Description: Constructor for the "diskSizeInclination" output analysis distribution operator operator class which takes a parameter set as input.
Code lines: 10
Contained by: file galacticus.output.analyses.distribution_operator.disk_size_inclination.F90
Modules used: input_parameters

function: disksizeinclinationoperatedistribution

Description: Implement a disk size inclination output analysis distribution operator.
Code lines: 16

Contained by: file galacticus.output.analyses.distribution_operator.disk_size_inclination.F90
Modules used: galacticus_error

function: disksizeinclinationoperatescalar
Description: Implement a disk size inclination output analysis distribution operator.
Code lines: 20
Contained by: file galacticus.output.analyses.distribution_operator.disk_size_inclination.F90

function: disksizeinclntnintegrandphi
Description: Integral for half-light radius.
Code lines: 7
Contained by: file galacticus.output.analyses.distribution_operator.disk_size_inclination.F90

function: disksizeinclntnintegrandx
Description: Integral for half-light radius.
Code lines: 15
Contained by: file galacticus.output.analyses.distribution_operator.disk_size_inclination.F90
Modules used: fgsl numerical_constants_math numerical_integration

function: disksizeinclntnroot
Description: Function used in solving for the half-light radii of inclined disks.
Code lines: 16
Contained by: file galacticus.output.analyses.distribution_operator.disk_size_inclination.F90
Modules used: fgsl numerical_constants_math numerical_integration

interface: outputanalysisdistributionoperatordisksizeinclntn
Description: Constructors for the "diskSizeInclination" output distribution operator class.
Code lines: 4
Contained by: file galacticus.output.analyses.distribution_operator.disk_size_inclination.F90

file: galacticus.output.analyses.distribution_operator.gravitational_lensing.F90
Description: Contains a module which implements a gravitational lensing output analysis distribution operator class.
Code lines: 228
Modules used: gravitational_lensing locks output_times

function: grvtnllnsngconstructorinternal
Description: Internal constructor for the "gravitational lensing" output analysis distribution operator class.
Code lines: 18
Contained by: file galacticus.output.analyses.distribution_operator.gravitational_lensing.F90
Modules used: iso_c_binding

function: grvtnllnsngconstructorparameters
Description: Constructor for the "gravitational lensing" output analysis distribution operator class which takes a parameter set as input.
Code lines: 27

Contained by: file galacticus.output.analyses.distribution_operator.gravitational_lensing.F90
Modules used: input_parameters

subroutine: grvtnllnsngdestructor
Description: Destructor for the "gravitational lensing" output analysis distribution operator class.
Code lines: 8
Contained by: file galacticus.output.analyses.distribution_operator.gravitational_lensing.F90

function: grvtnllnsngoperatedistribution
Description: Implement a gravitational lensing output analysis distribution operator.
Code lines: 90
Contained by: file galacticus.output.analyses.distribution_operator.gravitational_lensing.F90
Modules used: fgsl memory_management numerical_integration output_analyses_options

function: magnificationcdfintegrand
Description: Integrand over the gravitational lensing magnification cumulative distribution.
Code lines: 26
Contained by: function grvtnllnsngoperatedistribution
Modules used: galacticus_error output_analyses_options

function: grvtnllnsngoperatescalar
Description: Implement a gravitational lensing output analysis distribution operator.
Code lines: 16
Contained by: file galacticus.output.analyses.distribution_operator.gravitational_lensing.F90
Modules used: galacticus_error

type: grvtnllnsngtransfermatrix
Code lines: 2
Contained by: file galacticus.output.analyses.distribution_operator.gravitational_lensing.F90

interface: outputanalysisdistributionoperatorgrvtnllnsng
Description: Constructors for the "gravitational lensing" output analysis distribution operator class.
Code lines: 4
Contained by: file galacticus.output.analyses.distribution_operator.gravitational_lensing.F90

file: galacticus.output.analyses.distribution_operator.identity.F90
Description: Contains a module which implements a identity output analysis distribution operator class.
Code lines: 92

function: identityconstructorparameters
Description: Constructor for the "identity" output analysis distribution operator class which takes a parameter set as input.
Code lines: 10
Contained by: file galacticus.output.analyses.distribution_operator.identity.F90
Modules used: input_parameters

function: identityoperatedistribution
Description: Implement a identity output analysis distribution operator.

Code lines: 15
Contained by: file galacticus.output.analyses.distribution_operator.identity.F90
Modules used: galacticus_error

function: identityoperatescalar
Description: Implement a identity output analysis distribution operator.
Code lines: 23
Contained by: file galacticus.output.analyses.distribution_operator.identity.F90
Modules used: arrays_search

interface: outputanalysisdistributionoperatoridentity
Description: Constructors for the "identity" output analysis class.
Code lines: 3
Contained by: file galacticus.output.analyses.distribution_operator.identity.F90

file: galacticus.output.analyses.distribution_operator.random_error.F90
Description: Contains a module which implements a random error output analysis distribution operator class.
Code lines: 92

type: outputanalysisdistributionoperatorrandomerror
Description: A random error output distribution operator class.
Code lines: 16
Contained by: file galacticus.output.analyses.distribution_operator.random_error.F90

function: randomerroroperatedistribution
Description: Implement a random error output analysis distribution operator.
Code lines: 16
Contained by: file galacticus.output.analyses.distribution_operator.random_error.F90
Modules used: galacticus_error

function: randomerroroperatescalar
Description: Implement a random error output analysis distribution operator.
Code lines: 20
Contained by: file galacticus.output.analyses.distribution_operator.random_error.F90

file: galacticus.output.analyses.distribution_operator.random_error.HI_mass_ALFALFA.F90
Description: Contains a module which implements a random error output analysis distribution operator class providing errors in HI mass for the ALFALFA survey.
Code lines: 133
Modules used: output_analysis_molecular_ratios

interface: outputanalysisdistributionoperatorrandomerroralflf
Description: Constructors for the "randomErrorHIALFALFA" output analysis distribution operator class.
Code lines: 4
Contained by: file galacticus.output.analyses.distribution_operator.random_error.HI_mass_ALFALFA.F90

function: randomerrorhialfalfaconstructorinternal
Description: Internal constructor for the "randomErrorHIALFALFA" output analysis distribution operator class.

Code lines: 9
Contained by: file galacticus.output.analyses.distribution_operator.random_error.HI_mass_ALFALFA.F90

function: randomerrorhialfalfaconstructorparameters
Description: Constructor for the "randomErrorHIALFALFA" output analysis distribution operator class which takes a parameter set as input.
Code lines: 43
Contained by: file galacticus.output.analyses.distribution_operator.random_error.HI_mass_ALFALFA.F90
Modules used: input_parameters

subroutine: randomerrorhialfalfadestructor
Description: Destructor for the "randomErrorHIALFALFA" output analysis distribution operator class.
Code lines: 7
Contained by: file galacticus.output.analyses.distribution_operator.random_error.HI_mass_ALFALFA.F90

function: randomerrorhialfalfarootvariance
Description: Computes errors on $\log_{10}$(HI masses) for the ALFALFA survey analysis. Uses a simple fitting function. See constraints/dataAnalysis/hiMassFunction_ALFALFA_z0.00/alfalfaHIMassErrorModel.pl for details.
Code lines: 22
Contained by: file galacticus.output.analyses.distribution_operator.random_error.HI_mass_ALFALFA.F90

file: galacticus.output.analyses.distribution_operator.random_error.N-body_concentration.F90
Description: Contains a module which implements a random error output analysis distribution operator class providing errors in $\log_{10}$ of N-body halo concentration.
Code lines: 135
Modules used: node_property_extractors

interface: outputanalysisdistributionoperatorrndmerrnbdycnc
Description: Constructors for the "randomErrorNbdyCnc" output analysis distribution operator class.
Code lines: 4
Contained by: file galacticus.output.analyses.distribution_operator.random_error.N-body_concentration.F90

function: randomerrornbdycncconstructorinternal
Description: Internal constructor for the "randomErrorNbdyCnc" output analysis distribution operator class.
Code lines: 17
Contained by: file galacticus.output.analyses.distribution_operator.random_error.N-body_concentration.F90
Modules used: galacticus_error

function: randomerrornbdycncconstructorparameters
Description: Constructor for the "randomErrorNbdyCnc" output analysis distribution operator class which takes a parameter set as input.
Code lines: 37
Contained by: file galacticus.output.analyses.distribution_operator.random_error.N-body_concentration.F90
Modules used: input_parameters

subroutine: randomerrornbdycncdestructor
Description: Destructor for the "nbodyConcentration" output analysis distribution operator class.
Code lines: 6

Contained by: file galacticus.output.analyses.distribution_operator.random_error.N-body_concentration.F90

function: randomerrornbdycncrootvariance
Description: Computes errors on $\log_{10}$(halo concentration) for N-body halos.
Code lines: 22
Contained by: file galacticus.output.analyses.distribution_operator.random_error.N-body_concentration.F90

file: galacticus.output.analyses.distribution_operator.random_error.N-body_mass.F90
Description: Contains a module which implements a random error output analysis distribution operator class providing errors in $\log_{10}$ of N-body halo mass.
Code lines: 90
Modules used: statistics_nbody_halo_mass_errors

interface: outputanalysisdistributionoperatorrndmerrnbodymass
Description: Constructors for the "randomErrorNbodyMass" output analysis distribution operator class.
Code lines: 4
Contained by: file galacticus.output.analyses.distribution_operator.random_error.N-body_mass.F90

function: randomerrornbodymassconstructorinternal
Description: Internal constructor for the "randomErrorNbodyMass" output analysis distribution operator class.
Code lines: 8
Contained by: file galacticus.output.analyses.distribution_operator.random_error.N-body_mass.F90

function: randomerrornbodymassconstructorparameters
Description: Constructor for the "randomErrorNbodyMass" output analysis distribution operator class which takes a parameter set as input.
Code lines: 13
Contained by: file galacticus.output.analyses.distribution_operator.random_error.N-body_mass.F90
Modules used: input_parameters

subroutine: randomerrornbodymassdestructor
Description: Destructor for the "randomErrorNbodyMass" output analysis distribution operator class.
Code lines: 7
Contained by: file galacticus.output.analyses.distribution_operator.random_error.N-body_mass.F90

function: randomerrornbodymassrootvariance
Description: Computes errors on $\log_{10}$(halo masses) for N-body halos.
Code lines: 11
Contained by: file galacticus.output.analyses.distribution_operator.random_error.N-body_mass.F90

file: galacticus.output.analyses.distribution_operator.random_error.fixed.F90
Description: Contains a module which implements a random error output analysis distribution operator class.
Code lines: 83

interface: outputanalysisdistributionoperatorrandomerrorfixed
Description: Constructors for the "randomErrorFixed" output analysis distribution operator class.
Code lines: 4
Contained by: file galacticus.output.analyses.distribution_operator.random_error.fixed.F90

function: `randomerrorfixedconstructorinternal`
Description: Internal constructor for the "randomErrorFixed" output analysis distribution operator class.
Code lines: 8
Contained by: file `galacticus.output.analyses.distribution_operator.random_error.fixed.F90`

function: `randomerrorfixedconstructorparameters`
Description: Constructor for the "randomErrorFixed" output analysis distribution operator class which takes a parameter set as input.
Code lines: 21
Contained by: file `galacticus.output.analyses.distribution_operator.random_error.fixed.F90`
Modules used: `input_parameters`

function: `randomerrorfixedrootvariance`
Description: Return the root-variance in the fixed random error distribution operator.
Code lines: 10
Contained by: file `galacticus.output.analyses.distribution_operator.random_error.fixed.F90`

file: `galacticus.output.analyses.distribution_operator.random_error.polynomial.F90`
Description: Contains a module which implements a random error output analysis distribution operator class with an error magnitude that is a polynomial function of the property value.
Code lines: 119

interface: `outputanalysisdistributionoperatorrandomerrorplynml`
Description: Constructors for the "randomErrorPolynomial" output analysis distribution operator class.
Code lines: 4
Contained by: file `galacticus.output.analyses.distribution_operator.random_error.polynomial.F90`

function: `randomerrorpolynomialconstructorinternal`
Description: Internal constructor for the "randomErrorPolynomial" output analysis distribution operator class.
Code lines: 9
Contained by: file `galacticus.output.analyses.distribution_operator.random_error.polynomial.F90`

function: `randomerrorpolynomialconstructorparameters`
Description: Constructor for the "randomErrorPolynomial" output analysis distribution operator class which takes a parameter set as input.
Code lines: 47
Contained by: file `galacticus.output.analyses.distribution_operator.random_error.polynomial.F90`
Modules used: `input_parameters`

function: `randomerrorpolynomialrootvariance`
Description: Rerturn the root-variance in the polynomial random error distribution operator.
Code lines: 15
Contained by: file `galacticus.output.analyses.distribution_operator.random_error.polynomial.F90`

file: `galacticus.output.analyses.distribution_operator.sequence.F90`
Description: Contains a module which implements a sequence output analysis distribution operator class.
Code lines: 210

type: `distributionoperatorlist`
Code lines: 3
Contained by: file `galacticus.output.analyses.distribution_operator.sequence.F90`

interface: `outputanalysisdistributionoperatorsequence`
Description: Constructors for the "sequence" output analysis class.
Code lines: 4
Contained by: file `galacticus.output.analyses.distribution_operator.sequence.F90`

function: `sequenceconstructorinternal`
Description: Internal constructor for the sequence merger tree normalizer class.
Code lines: 14
Contained by: file `galacticus.output.analyses.distribution_operator.sequence.F90`

function: `sequenceconstructorparameters`
Description: Constructor for the "sequence" output analysis distribution operator class which takes a parameter set as input.
Code lines: 22
Contained by: file `galacticus.output.analyses.distribution_operator.sequence.F90`
Modules used: `input_parameters`

subroutine: `sequencedeepcopy`
Description: Perform a deep copy for the `sequence` output analysis distribution operator class.
Code lines: 31
Contained by: file `galacticus.output.analyses.distribution_operator.sequence.F90`
Modules used: `galacticus_error`

subroutine: `sequencedestructor`
Description: Destructor for the sequence distribution operator class.
Code lines: 16
Contained by: file `galacticus.output.analyses.distribution_operator.sequence.F90`

function: `sequenceoperatedistribution`
Description: Implement a random error output analysis distribution operator.
Code lines: 20
Contained by: file `galacticus.output.analyses.distribution_operator.sequence.F90`
Modules used: `galacticus_error`

function: `sequenceoperatescalar`
Description: Implement an sequence output analysis distribution operator.
Code lines: 23
Contained by: file `galacticus.output.analyses.distribution_operator.sequence.F90`

subroutine: `sequenceprepend`
Description: Prepend an operator to the sequence.
Code lines: 12
Contained by: file `galacticus.output.analyses.distribution_operator.sequence.F90`

file: galacticus.output.analyses.distribution_operator.spin_N-body_errors.F90
Description: Contains a module which implements an output analysis distribution operator class to account for errors on N-body measurements of halo spin.
Code lines: 169
Modules used: halo_spin_distributions

interface: outputanalysisdistributionoperatorspinnbodyerrors
Description: Constructors for the "spinNBodyErrors" output analysis distribution operator class.
Code lines: 4
Contained by: file galacticus.output.analyses.distribution_operator.spin_N-body_errors.F90

function: spinnbodyerrorsconstructorinternal
Description: Internal constructor for the "spinNBodyErrors" output analysis distribution operator class.
Code lines: 17
Contained by: file galacticus.output.analyses.distribution_operator.spin_N-body_errors.F90
Modules used: galacticus_error

function: spinnbodyerrorsconstructorparameters
Description: Constructor for the "spinNBodyErrors" output analysis distribution operator class which takes a parameter set as input.
Code lines: 14
Contained by: file galacticus.output.analyses.distribution_operator.spin_N-body_errors.F90
Modules used: input_parameters

subroutine: spinnbodyerrorsdestructor
Description: Destructor for the "spinNBodyErrors" output analysis distribution operator class.
Code lines: 7
Contained by: file galacticus.output.analyses.distribution_operator.spin_N-body_errors.F90

function: spinnbodyerrorsoperatedistribution
Description: Implement an output analysis distribution operator which accounts for errors in N-body measurements of halo spin.
Code lines: 16
Contained by: file galacticus.output.analyses.distribution_operator.spin_N-body_errors.F90
Modules used: galacticus_error

function: spinnbodyerrorsoperatescalar
Description: Implement an output analysis distribution operator which accounts for errors in N-body measurements of halo spin.
Code lines: 62
Contained by: file galacticus.output.analyses.distribution_operator.spin_N-body_errors.F90
Modules used: fgsl galacticus_error numerical_integration output_analyses_options

function: spindistributionintegrate
Description: Integrand function used to find cumulative spin distribution over a bin.
Code lines: 12
Contained by: function spinnbodyerrorsoperatescalar

file: galacticus.output.analyses.galaxy_sizes_SDSS.F90
Description: Contains a module which implements a galaxy size output analysis class for SDSS data.
Code lines: 334
Modules used: cosmology_functions

function: galaxysizessdssconstructorinternal
Description: Internal constructor for the "galaxySizesSDSS" output analysis class.
Code lines: 245
Contained by: file galacticus.output.analyses.galaxy_sizes_SDSS.F90
Modules used: cosmology_parameters, galacticus_error, galacticus_paths, io_hdf5, iso_varying_string, memory_management, numerical_comparison, numerical_constants_astronomical, numerical_constants_prefixes, output_analyses_options, output_analysis_utilities, output_times

function: galaxysizessdssconstructorparameters
Description: Constructor for the "galaxySizesSDSS" output analysis class which takes a parameter set as input.
Code lines: 33
Contained by: file galacticus.output.analyses.galaxy_sizes_SDSS.F90
Modules used: input_parameters

subroutine: galaxysizessdssdestructor
Description: Destructor for the "galaxySizesSDSS" output analysis class.
Code lines: 7
Contained by: file galacticus.output.analyses.galaxy_sizes_SDSS.F90

interface: outputanalysisgalaxysizessdss
Description: Constructors for the "galaxySizesSDSS" output analysis class.
Code lines: 4
Contained by: file galacticus.output.analyses.galaxy_sizes_SDSS.F90

file: galacticus.output.analyses.luminosity_function.F90
Description: Contains a module which implements a luminosity function output analysis class.
Code lines: 334
Modules used: cosmology_functions, geometry_surveys

function: luminosityfunctionconstructorfile
Description: Constructor for the "luminosityFunction" output analysis class which reads bin information from a standard format file.
Code lines: 49
Contained by: file galacticus.output.analyses.luminosity_function.F90
Modules used: io_hdf5

function: luminosityfunctionconstructorinternal
Description: Constructor for the "luminosityFunction" output analysis class which takes a parameter set as input.
Code lines: 94

Contained by: file galacticus.output.analyses.luminosity_function.F90
Modules used: galacticus_error, iso_varying_string, memory_management, numerical_constants_astronomical, output_analyses_options, output_analysis_utilities, string_handling

function: luminosityfunctionconstructorparameters
Description: Constructor for the "luminosityFunction" output analysis class which takes a parameter set as input.
Code lines: 130
Contained by: file galacticus.output.analyses.luminosity_function.F90
Modules used: input_parameters

subroutine: luminosityfunctiondestructor
Description: Destructor for the "luminosityFunction" output analysis class.
Code lines: 8
Contained by: file galacticus.output.analyses.luminosity_function.F90

interface: outputanalysisluminosityfunction
Description: Constructors for the "luminosityFunction" output analysis class.
Code lines: 5
Contained by: file galacticus.output.analyses.luminosity_function.F90

file: galacticus.output.analyses.luminosity_function.Halpha.F90
Description: Contains a module which implements a luminosity function output analysis class.
Code lines: 381
Modules used: cosmology_functions, geometry_surveys, stellar_spectra_dust_attenuations

function: luminosityfunctionhalphaconstructorfile
Description: Constructor for the "luminosityFunctionHalpha" output analysis class which reads bin information from a standard format file.
Code lines: 50
Contained by: file galacticus.output.analyses.luminosity_function.Halpha.F90
Modules used: io_hdf5

function: luminosityfunctionhalphaconstructorinternal
Description: Constructor for the "luminosityFunctionHalpha" output analysis class which takes a parameter set as input.
Code lines: 120
Contained by: file galacticus.output.analyses.luminosity_function.Halpha.F90
Modules used: galacticus_error, iso_varying_string, memory_management, numerical_constants_astronomical, numerical_constants_units, output_analyses_options, output_analysis_utilities, output_times, string_handling

function: luminosityfunctionhalphaconstructorparameters
Description: Constructor for the "luminosityFunctionHalpha" output analysis class which takes a parameter set as input.

Code lines: 147
Contained by: file galacticus.output.analyses.luminosity_function.Halpha.F90
Modules used: input_parameters

subroutine: luminosityfunctionhalphadestructor
Description: Destructor for the "luminosityFunctionHalpha" output analysis class.
Code lines: 9
Contained by: file galacticus.output.analyses.luminosity_function.Halpha.F90

interface: outputanalysisluminosityfunctionhalpha
Description: Constructors for the "luminosityFunctionHalpha" output analysis class.
Code lines: 5
Contained by: file galacticus.output.analyses.luminosity_function.Halpha.F90

file: galacticus.output.analyses.luminosity_function.Halpha.Gunawardhana2013.SDSS.F90
Description: Contains a module which implements a stellar mass function output analysis class.
Code lines: 266
Modules used: gravitational_lensing

function: luminosityfunctiongunawardhana2013sdssconstructorinternal
Description: Constructor for the "luminosityFunctionGunawardhana2013SDSS" output analysis class for internal use.
Code lines: 107
Contained by: file galacticus.output.analyses.luminosity_function.Halpha.Gunawardhana2013.SDSS.F90
Modules used: cosmology_parameters, galacticus_error, galacticus_paths, input_parameters, output_analysis_distribution_operators

function: luminosityfunctiongunawardhana2013sdssconstructorparameters
Description: Constructor for the "luminosityFunctionGunawardhana2013SDSS" output analysis class which takes a parameter set as input.
Code lines: 117
Contained by: file galacticus.output.analyses.luminosity_function.Halpha.Gunawardhana2013.SDSS.F90
Modules used: input_parameters

interface: outputanalysisluminosityfunctiongunawardhana2013sdss
Description: Constructors for the "luminosityFunctionGunawardhana2013SDSS" output analysis class.
Code lines: 4
Contained by: file galacticus.output.analyses.luminosity_function.Halpha.Gunawardhana2013.SDSS.F90

file: galacticus.output.analyses.luminosity_function.Halpha.Sobral2013.HiZELS.F90
Description: Contains a module which implements a stellar mass function output analysis class.
Code lines: 292
Modules used: gravitational_lensing

function: luminosityfunctionsobral2013hizelsconstructorinternal
Description: Constructor for the "luminosityFunctionSobral2013HiZELS" output analysis class for internal use.
Code lines: 125

Contained by: file galacticus.output.analyses.luminosity_function.Halpha.Sobral2013.HiZELS.F90
Modules used: cosmology_parameters, galacticus_error, galacticus_paths, input_parameters, output_analysis_distribution_operators, string_handling

function: luminosityfunctionsobral2013hizelsconstructorparameters
Description: Constructor for the "luminosityFunctionSobral2013HiZELS" output analysis class which takes a parameter set as input.
Code lines: 125
Contained by: file galacticus.output.analyses.luminosity_function.Halpha.Sobral2013.HiZELS.F90
Modules used: input_parameters

interface: outputanalysisluminosityfunctionsobral2013hizels
Description: Constructors for the "luminosityFunctionSobral2013HiZELS" output analysis class.
Code lines: 4
Contained by: file galacticus.output.analyses.luminosity_function.Halpha.Sobral2013.HiZELS.F90

file: galacticus.output.analyses.luminosity_function.Montero-Dorta2009.SDSS.F90
Description: Contains a module which implements a stellar mass function output analysis class.
Code lines: 282
Modules used: gravitational_lensing

function: luminosityfunctionmonterodorta2009sdssconstructorinternal
Description: Constructor for the "luminosityFunctionMonteroDorta2009SDSS" output analysis class for internal use.
Code lines: 122
Contained by: file galacticus.output.analyses.luminosity_function.Montero-Dorta2009.SDSS.F90
Modules used: cosmology_parameters, galacticus_error, galacticus_paths, input_parameters, output_analysis_distribution_operators

function: luminosityfunctionmonterodorta2009sdssconstructorparameters
Description: Constructor for the "luminosityFunctionMonteroDorta2009SDSS" output analysis class which takes a parameter set as input.
Code lines: 117
Contained by: file galacticus.output.analyses.luminosity_function.Montero-Dorta2009.SDSS.F90
Modules used: input_parameters

interface: outputanalysisluminosityfunctionmonterodorta2009sdss
Description: Constructors for the "luminosityFunctionMonteroDorta2009SDSS" output analysis class.
Code lines: 4
Contained by: file galacticus.output.analyses.luminosity_function.Montero-Dorta2009.SDSS.F90

file: galacticus.output.analyses.mass_function_HI.ALFALFA_Martin2010.F90
Description: Contains a module which implements an ALFALFA HI mass function output analysis class.
Code lines: 222
Modules used: gravitational_lensing

function: `massfunctionhialfalfamartin2010constructorinternal`

Description: Constructor for the "massFunctionHIALFALFAMartin2010" output analysis class for internal use.
Code lines: 95
Contained by: file `galacticus.output.analyses.mass_function_HI.ALFALFA_Martin2010.F90`
Modules used: `cosmology_parameters`, `galacticus_paths`, `input_parameters`, `output_analysis_distribution_operators`, `output_analysis_molecular_ratios`

function: `massfunctionhialfalfamartin2010constructorparameters`

Description: Constructor for the "massFunctionHIALFALFAMartin2010" output analysis class which takes a parameter set as input.
Code lines: 85
Contained by: file `galacticus.output.analyses.mass_function_HI.ALFALFA_Martin2010.F90`
Modules used: `cosmology_parameters`, `input_parameters`, `output_analysis_molecular_ratios`

interface: `outputanalysismassfunctionhialfalfamartin2010`

Description: Constructors for the "massFunctionHIALFALFAMartin2010" output analysis class.
Code lines: 4
Contained by: file `galacticus.output.analyses.mass_function_HI.ALFALFA_Martin2010.F90`

file: `galacticus.output.analyses.mass_function_HI.F90`

Description: Contains a module which implements an HI mass function output analysis class.
Code lines: 353
Modules used: `cosmology_functions`, `geometry_surveys`

function: `massfunctionhiconstructorfile`

Description: Constructor for the "massFunctionHI" output analysis class which reads bin information from a standard format file.
Code lines: 42
Contained by: file `galacticus.output.analyses.mass_function_HI.F90`
Modules used: `io_hdf5`, `output_analysis_molecular_ratios`

function: `massfunctionhiconstructorinternal`

Description: Constructor for the "massFunctionHI" output analysis class which takes a parameter set as input.
Code lines: 117
Contained by: file `galacticus.output.analyses.mass_function_HI.F90`
Modules used: `galacticus_error`, `iso_varying_string`, `memory_management`, `numerical_constants_astronomical`, `output_analyses_options`, `output_analysis_molecular_ratios`, `output_analysis_utilities`, `output_times`, `string_handling`

function: `massfunctionhiconstructorparameters`

Description: Constructor for the "massFunctionHI" output analysis class which takes a parameter set as input.
Code lines: 134

Contained by: file galacticus.output.analyses.mass_function_HI.F90
Modules used: input_parameters output_analysis_molecular_ratios

subroutine: massfunctionhidestructor
Description: Destructor for the "massFunctionHI" output analysis class.
Code lines: 8
Contained by: file galacticus.output.analyses.mass_function_HI.F90

interface: outputanalysismassfunctionhi
Description: Constructors for the "massFunctionHI" output analysis class.
Code lines: 5
Contained by: file galacticus.output.analyses.mass_function_HI.F90

file: galacticus.output.analyses.mass_function_stellar.Bernardi_SDSS.F90
Description: Contains a module which implements an output analyis class for the Bernardi et al. [2013] stellar mass function.
Code lines: 253
Modules used: gravitational_lensing

function: massfunctionstellarbernardi2013sdssconstructorinternal
Description: Constructor for the "massFunctionStellarBernardi2013SDSS" output analysis class for internal use.
Code lines: 105
Contained by: file galacticus.output.analyses.mass_function_stellar.Bernardi_SDSS.F90
Modules used: cosmology_parameters galacticus_paths input_parameters output_analysis_distribution_operators

function: massfunctionstellarbernardi2013sdssconstructorparameters
Description: Constructor for the "massFunctionStellarBernardi2013SDSS" output analysis class which takes a parameter set as input.
Code lines: 106
Contained by: file galacticus.output.analyses.mass_function_stellar.Bernardi_SDSS.F90
Modules used: input_parameters

interface: outputanalysismassfunctionstellarbernardi2013sdss
Description: Constructors for the "massFunctionStellarBernardi2013SDSS" output analysis class.
Code lines: 4
Contained by: file galacticus.output.analyses.mass_function_stellar.Bernardi_SDSS.F90

file: galacticus.output.analyses.mass_function_stellar.F90
Description: Contains a module which implements a stellar mass function output analysis class.
Code lines: 339
Modules used: cosmology_functions geometry_surveys

function: massfunctionstellarconstructorfile
Description: Constructor for the "massFunctionStellar" output analysis class which reads bin information from a standard format file.
Code lines: 40
Contained by: file galacticus.output.analyses.mass_function_stellar.F90
Modules used: io_hdf5

function: `massfunctionstellarconstructorinternal`

Description: Constructor for the "massFunctionStellar" output analysis class which takes a parameter set as input.
Code lines: 108
Contained by: file `galacticus.output.analyses.mass_function_stellar.F90`
Modules used: `galacticus_error` `iso_varying_string` `memory_management` `numerical_constants_astronomical` `output_analyses_options` `output_analysis_utilities` `string_handling`

function: `massfunctionstellarconstructorparameters`

Description: Constructor for the "massFunctionStellar" output analysis class which takes a parameter set as input.
Code lines: 131
Contained by: file `galacticus.output.analyses.mass_function_stellar.F90`
Modules used: `galacticus_error` `input_parameters`

subroutine: `massfunctionstellardestructor`

Description: Destructor for the "massFunctionStellar" output analysis class.
Code lines: 8
Contained by: file `galacticus.output.analyses.mass_function_stellar.F90`

interface: `outputanalysismassfunctionstellar`

Description: Constructors for the "massFunctionStellar" output analysis class.
Code lines: 5
Contained by: file `galacticus.output.analyses.mass_function_stellar.F90`

file: `galacticus.output.analyses.mass_function_stellar.GAMA.F90`

Description: Contains a module which implements an output analysis class for the Baldry et al. [2012] stellar mass function.
Code lines: 253
Modules used: `gravitational_lensing`

function: `massfunctionstellarbaldry2012gamaconstructorinternal`

Description: Constructor for the "massFunctionStellarBaldry2012GAMA" output analysis class for internal use.
Code lines: 105
Contained by: file `galacticus.output.analyses.mass_function_stellar.GAMA.F90`
Modules used: `cosmology_parameters` `galacticus_paths` `input_parameters` `output_analysis_distribution_operators`

function: `massfunctionstellarbaldry2012gamaconstructorparameters`

Description: Constructor for the "massFunctionStellarBaldry2012GAMA" output analysis class which takes a parameter set as input.
Code lines: 106
Contained by: file `galacticus.output.analyses.mass_function_stellar.GAMA.F90`
Modules used: `input_parameters`

interface: `outputanalysismassfunctionstellarbaldry2012gama`
Description: Constructors for the "massFunctionStellarBaldry2012GAMA" output analysis class.
Code lines: 4
Contained by: file `galacticus.output.analyses.mass_function_stellar.GAMA.F90`

file: `galacticus.output.analyses.mass_function_stellar.PRIMUS.F90`
Description: Contains a module which implements a stellar mass function output analysis class for the PRIMUS survey of Moustakas et al. [2013].
Code lines: 290
Modules used: `gravitational_lensing`

function: `massfunctionstellarprimusconstructorinternal`
Description: Constructor for the "massFunctionStellarPRIMUS" output analysis class for internal use.
Code lines: 134
Contained by: file `galacticus.output.analyses.mass_function_stellar.PRIMUS.F90`
Modules used: `cosmology_parameters`, `galacticus_error`, `galacticus_paths`, `input_parameters`, `iso_varying_string`, `output_analysis_distribution_operators`, `string_handling`

function: `massfunctionstellarprimusconstructorparameters`
Description: Constructor for the "massFunctionStellarPRIMUS" output analysis class which takes a parameter set as input.
Code lines: 114
Contained by: file `galacticus.output.analyses.mass_function_stellar.PRIMUS.F90`
Modules used: `input_parameters`

interface: `outputanalysismassfunctionstellarprimus`
Description: Constructors for the "massFunctionStellarPRIMUS" output analysis class.
Code lines: 4
Contained by: file `galacticus.output.analyses.mass_function_stellar.PRIMUS.F90`

file: `galacticus.output.analyses.mass_function_stellar.SDSS.F90`
Description: Contains a module which implements a stellar mass function output analysis class.
Code lines: 253
Modules used: `gravitational_lensing`

function: `massfunctionstellarsdssconstructorinternal`
Description: Constructor for the "massFunctionStellarSDSS" output analysis class for internal use.
Code lines: 105
Contained by: file `galacticus.output.analyses.mass_function_stellar.SDSS.F90`
Modules used: `cosmology_parameters`, `galacticus_paths`, `input_parameters`, `output_analysis_distribution_operators`

function: `massfunctionstellarsdssconstructorparameters`
Description: Constructor for the "massFunctionStellarSDSS" output analysis class which takes a parameter set as input.

Code lines: 106
Contained by: file galacticus.output.analyses.mass_function_stellar.SDSS.F90
Modules used: input_parameters

interface: outputanalysismassfunctionstellarsdss
Description: Constructors for the "massFunctionStellarSDSS" output analysis class.
Code lines: 4
Contained by: file galacticus.output.analyses.mass_function_stellar.SDSS.F90

file: galacticus.output.analyses.mass_function_stellar.UKIDSS_UDS.F90
Description: Contains a module which implements a stellar mass function output analysis class for the UKIDSS UDS survey of Caputi et al. [2011].
Code lines: 281
Modules used: gravitational_lensing

function: massfunctionstellarukidssudsconstructorinternal
Description: Constructor for the "massFunctionStellarUKIDSSUDS" output analysis class for internal use.
Code lines: 125
Contained by: file galacticus.output.analyses.mass_function_stellar.UKIDSS_UDS.F90
Modules used: cosmology_parameters, galacticus_error, galacticus_paths, input_parameters, iso_varying_string, output_analysis_distribution_operators, string_handling

function: massfunctionstellarukidssudsconstructorparameters
Description: Constructor for the "massFunctionStellarUKIDSSUDS" output analysis class which takes a parameter set as input.
Code lines: 114
Contained by: file galacticus.output.analyses.mass_function_stellar.UKIDSS_UDS.F90
Modules used: input_parameters

interface: outputanalysismassfunctionstellarukidssuds
Description: Constructors for the "massFunctionStellarUKIDSSUDS" output analysis class.
Code lines: 4
Contained by: file galacticus.output.analyses.mass_function_stellar.UKIDSS_UDS.F90

file: galacticus.output.analyses.mass_function_stellar.ULTRAVISTA.F90
Description: Contains a module which implements a stellar mass function output analysis class for the ULTRAVISTA survey of Muzzin et al. [2013].
Code lines: 293
Modules used: gravitational_lensing

function: massfunctionstellarultravistaconstructorinternal
Description: Constructor for the "massFunctionStellarULTRAVISTA" output analysis class for internal use.
Code lines: 137
Contained by: file galacticus.output.analyses.mass_function_stellar.ULTRAVISTA.F90
Modules used: cosmology_parameters, galacticus_error

galacticus_paths
iso_varying_string
string_handling
input_parameters
output_analysis_distribution_operators

function: massfunctionstellarultravistaconstructorparameters
Description: Constructor for the "massFunctionStellarULTRAVISTA" output analysis class which takes a parameter set as input.
Code lines: 114
Contained by: file galacticus.output.analyses.mass_function_stellar.ULTRAVISTA.F90
Modules used: input_parameters

interface: outputanalysismassfunctionstellarultravista
Description: Constructors for the "massFunctionStellarULTRAVISTA" output analysis class.
Code lines: 4
Contained by: file galacticus.output.analyses.mass_function_stellar.ULTRAVISTA.F90

file: galacticus.output.analyses.mass_function_stellar.VIPERS.F90
Description: Contains a module which implements a stellar mass function output analysis class for the VIPERS survey of Davidzon et al. [2013].
Code lines: 281
Modules used: gravitational_lensing

function: massfunctionstellarvipersconstructorinternal
Description: Constructor for the "massFunctionStellarVIPERS" output analysis class for internal use.
Code lines: 125
Contained by: file galacticus.output.analyses.mass_function_stellar.VIPERS.F90
Modules used: cosmology_parameters, galacticus_error, galacticus_paths, input_parameters, iso_varying_string, output_analysis_distribution_operators, string_handling

function: massfunctionstellarvipersconstructorparameters
Description: Constructor for the "massFunctionStellarVIPERS" output analysis class which takes a parameter set as input.
Code lines: 114
Contained by: file galacticus.output.analyses.mass_function_stellar.VIPERS.F90
Modules used: input_parameters

interface: outputanalysismassfunctionstellarvipers
Description: Constructors for the "massFunctionStellarVIPERS" output analysis class.
Code lines: 4
Contained by: file galacticus.output.analyses.mass_function_stellar.VIPERS.F90

file: galacticus.output.analyses.mass_function_stellar.ZFOURGE.F90
Description: Contains a module which implements a stellar mass function output analysis class for the ZFOURGE survey of Tomczak et al. [2014].
Code lines: 296
Modules used: gravitational_lensing

function: massfunctionstellarzfourgeconstructorinternal
Description: Constructor for the "massFunctionStellarZFOURGE" output analysis class for internal use.
Code lines: 140
Contained by: file galacticus.output.analyses.mass_function_stellar.ZFOURGE.F90
Modules used: cosmology_parameters, galacticus_error, galacticus_paths, input_parameters, iso_varying_string, output_analysis_distribution_operators, string_handling

function: massfunctionstellarzfourgeconstructorparameters
Description: Constructor for the "massFunctionStellarZFOURGE" output analysis class which takes a parameter set as input.
Code lines: 114
Contained by: file galacticus.output.analyses.mass_function_stellar.ZFOURGE.F90
Modules used: input_parameters

interface: outputanalysismassfunctionstellarzfourge
Description: Constructors for the "massFunctionStellarZFOURGE" output analysis class.
Code lines: 4
Contained by: file galacticus.output.analyses.mass_function_stellar.ZFOURGE.F90

file: galacticus.output.analyses.mass_metallicity_relation.Andrews2013.F90
Description: Contains a module which implements a mass-metallicity relation analysis class.
Code lines: 319

function: massmetallicityandrews2013constructorinternal
Description: Constructor for the "massMetallicityAndrews2013" output analysis class for internal use.
Code lines: 208
Contained by: file galacticus.output.analyses.mass_metallicity_relation.Andrews2013.F90
Modules used: abundances_structure, atomic_data, cosmology_functions, cosmology_parameters, galacticus_error, galacticus_paths, io_hdf5, memory_management, node_property_extractors, numerical_constants_astronomical, output_analysis_distribution_operators, output_analysis_property_operators, output_analysis_utilities, output_analysis_weight_operators, output_times

function: massmetallicityandrews2013constructorparameters
Description: Constructor for the "massMetallicityAndrews2013" output analysis class which takes a parameter set as input.
Code lines: 70
Contained by: file galacticus.output.analyses.mass_metallicity_relation.Andrews2013.F90
Modules used: cosmology_functions, input_parameters

interface: outputanalysismassmetallicityandrews2013

Description: Constructors for the "massMetallicityAndrews2013" output analysis class.
Code lines: 4
Contained by: file galacticus.output.analyses.mass_metallicity_relation.Andrews2013.F90

file: galacticus.output.analyses.mass_metallicity_relation.Blanc2017.F90
Description: Contains a module which implements a mass-metallicity relation analysis class.
Code lines: 323

function: massmetallicityblanc2017constructorinternal
Description: Constructor for the "massMetallicityBlanc2017" output analysis class for internal use.
Code lines: 212
Contained by: file galacticus.output.analyses.mass_metallicity_relation.Blanc2017.F90
Modules used: abundances_structure, atomic_data, cosmology_functions, cosmology_parameters, galacticus_error, galacticus_paths, io_hdf5, memory_management, node_property_extractors, numerical_constants_astronomical, output_analysis_distribution_operators, output_analysis_property_operators, output_analysis_utilities, output_analysis_weight_operators, output_times

function: massmetallicityblanc2017constructorparameters
Description: Constructor for the "massMetallicityBlanc2017" output analysis class which takes a parameter set as input.
Code lines: 70
Contained by: file galacticus.output.analyses.mass_metallicity_relation.Blanc2017.F90
Modules used: cosmology_functions, input_parameters

interface: outputanalysismassmetallicityblanc2017
Description: Constructors for the "massMetallicityBlanc2017" output analysis class.
Code lines: 4
Contained by: file galacticus.output.analyses.mass_metallicity_relation.Blanc2017.F90

file: galacticus.output.analyses.mean_function_1d.F90
Description: Contains a module which implements a generic 1D mean function (i.e. mean value of some property weighted by number density of objects binned by some property) output analysis class.
Code lines: 812
Modules used: galactic_filters, iso_c_binding, iso_varying_string, node_property_extractors, output_analyses_options, output_analysis_distribution_normalizers, output_analysis_distribution_operators, output_analysis_property_operators, output_analysis_weight_operators, output_times

subroutine: meanfunction1danalyze
Description: Implement a meanFunction1D output analysis.

Code lines: 12
Contained by: file galacticus.output.analyses.mean_function_1d.F90

function: meanfunction1dconstructorinternal
Description: Constructor for the "meanFunction1D" output analysis class for internal use.
Code lines: 226
Contained by: file galacticus.output.analyses.mean_function_1d.F90
Modules used: memory_management

function: meanfunction1dconstructorparameters
Description: Constructor for the "meanFunction1D" output analysis class which takes a parameter set as input.
Code lines: 300
Contained by: file galacticus.output.analyses.mean_function_1d.F90
Modules used: galacticus_error input_parameters memory_management

subroutine: meanfunction1ddestructor
Description: Destructor for the meanFunction1D output analysis class.
Code lines: 9
Contained by: file galacticus.output.analyses.mean_function_1d.F90

subroutine: meanfunction1dfinalize
Description: Implement a meanFunction1D output analysis finalization.
Code lines: 56
Contained by: file galacticus.output.analyses.mean_function_1d.F90
Modules used: galacticus_hdf5 io_hdf5

subroutine: meanfunction1dfinalizeanalysis
Description: Finalize analysis of a meanFunction1D output analysis.
Code lines: 34
Contained by: file galacticus.output.analyses.mean_function_1d.F90

function: meanfunction1dloglikelihood
Description: Return the log-likelihood of a meanFunction1D output analysis.
Code lines: 32
Contained by: file galacticus.output.analyses.mean_function_1d.F90
Modules used: galacticus_error linear_algebra numerical_constants_math

subroutine: meanfunction1dreduce
Description: Implement a volumeFunction1D output analysis reduction.
Code lines: 16
Contained by: file galacticus.output.analyses.mean_function_1d.F90
Modules used: galacticus_error

subroutine: meanfunction1dresults
Description: Implement a meanFunction1D output analysis finalization.
Code lines: 27

Contained by: file galacticus.output.analyses.mean_function_1d.F90
Modules used: memory_management

interface: outputanalysismeanfunction1d
Description: Constructors for the "meanFunction1D" output analysis class.
Code lines: 4
Contained by: file galacticus.output.analyses.mean_function_1d.F90

file: galacticus.output.analyses.molecular_ratio.F90
Description: Contains a module which provides a class that implements on-the-fly analyses.
Code lines: 49

module: output_analysis_molecular_ratios
Description: Provides a class that implements operators on properties for on-the-fly analyses.
Code lines: 27
Contained by: file galacticus.output.analyses.molecular_ratio.F90
Modules used: galacticus_nodes, iso_c_binding
Used by: subroutine galacticus_function_classes_destroy; function hivshalomassrelationpadmanabhan2017constructorin; function hivshalomassrelationpadmanabhan2017constructorpa; file galacticus.output.analyses.distribution_operator.random_error.HI_mass_ALFALFA.F90; function massfunctionhialfalfamartin2010constructorfile; function massfunctionhialfalfamartin2010constructorinternal; function massfunctionhialfalfamartin2010constructorparame; function massfunctionhiconstructorfile; function massfunctionhiconstructorinternal; function massfunctionhiconstructorparameters; file galacticus.output.analyses.property_operator.HI_mass.F90; subroutine galacticus_state_retrieve; subroutine galacticus_state_store

file: galacticus.output.analyses.molecular_ratio.Obreschkow2009.F90
Description: Contains a module which implements a molecular ratio class that assumes the model of Obreschkow et al. [2009].
Code lines: 181

function: obreschkow2009constructorinternal
Description: Internal constructor for the "obreschkow2009" output analysis distribution operator class.
Code lines: 9
Contained by: file galacticus.output.analyses.molecular_ratio.Obreschkow2009.F90
Modules used: input_parameters

function: obreschkow2009constructorparameters
Description: Constructor for the "obreschkow2009" output analysis property operator class which takes a parameter set as input.
Code lines: 86
Contained by: file galacticus.output.analyses.molecular_ratio.Obreschkow2009.F90
Modules used: input_parameters

function: `obreschkow2009ratio`
Description: Compute the molecular fraction in the `obreschkow2009` class.
Code lines: 28
Contained by: file `galacticus.output.analyses.molecular_ratio.Obreschkow2009.F90`
Modules used: `galacticus_nodes` `iso_c_binding` `numerical_constants_astronomical`

function: `obreschkow2009ratioscatter`
Description: Compute the scatter in molecular fraction in the `obreschkow2009` class.
Code lines: 10
Contained by: file `galacticus.output.analyses.molecular_ratio.Obreschkow2009.F90`

interface: `outputanalysismolecularratioobreschkow2009`
Description: Constructors for the "obreschkow2009" output analysis class.
Code lines: 4
Contained by: file `galacticus.output.analyses.molecular_ratio.Obreschkow2009.F90`

file: `galacticus.output.analyses.morphological_fraction.GAMA_Moffett2016.F90`
Description: Contains a module which implements a stellar vs halo mass relation analysis class.
Code lines: 430

function: `morphologicalfractiongamamoffett2016constructorinternal`
Description: Constructor for the "morphologicalFractionGAMAMoffett2016" output analysis class for internal use.
Code lines: 211
Contained by: file `galacticus.output.analyses.morphological_fraction.GAMA_Moffett2016.F90`
Modules used: `cosmology_functions` `cosmology_parameters` `galacticus_paths` `io_hdf5` `memory_management` `node_property_extractors` `numerical_constants_astronomical` `output_analysis_distribution_operators` `output_analysis_property_operators` `output_analysis_utilities` `output_analysis_weight_operators` `output_times` `statistics_distributions`

function: `morphologicalfractiongamamoffett2016constructorparameters`
Description: Constructor for the "morphologicalFractionGAMAMoffett2016" output analysis class which takes a parameter set as input.
Code lines: 76
Contained by: file `galacticus.output.analyses.morphological_fraction.GAMA_Moffett2016.F90`
Modules used: `cosmology_functions` `input_parameters`

subroutine: `morphologicalfractiongamamoffett2016finalize`
Description: Implement a `morphologicalFractionGAMAMoffett2016` output analysis finalization.
Code lines: 59
Contained by: file `galacticus.output.analyses.morphological_fraction.GAMA_Moffett2016.F90`
Modules used: `galacticus_hdf5` `io_hdf5`

function: `morphologicalfractiongamamoffett2016loglikelihood`
Description: Return the log-likelihood of a morphologicalFractionGAMAMoffett2016 output analysis.
Code lines: 35
Contained by: file galacticus.output.analyses.morphological_fraction.GAMA_Moffett2016.F90
Modules used: models_likelihoods_constants

interface: `outputanalysismorphologicalfractiongamamoffett2016`
Description: Constructors for the "morphologicalFractionGAMAMoffett2016" output analysis class.
Code lines: 4
Contained by: file galacticus.output.analyses.morphological_fraction.GAMA_Moffett2016.F90

file: `galacticus.output.analyses.multi.F90`
Description: Implements a merger trees analysis class which combines multiple other analyses.
Code lines: 208

type: `multianalysislist`
Code lines: 3
Contained by: file galacticus.output.analyses.multi.F90

subroutine: `multianalyze`
Description: Output from all analyses.
Code lines: 14
Contained by: file galacticus.output.analyses.multi.F90

function: `multiconstructorinternal`
Description: Internal constructor for the `multi` analysis class.
Code lines: 14
Contained by: file galacticus.output.analyses.multi.F90

function: `multiconstructorparameters`
Description: Constructor for the `multi` merger tree analysis class which takes a parameter set as input.
Code lines: 22
Contained by: file galacticus.output.analyses.multi.F90
Modules used: input_parameters

subroutine: `multideepcopy`
Description: Perform a deep copy for the `multi` analysis class.
Code lines: 31
Contained by: file galacticus.output.analyses.multi.F90
Modules used: galacticus_error

subroutine: `multidestructor`
Description: Destructor for the `multi` analysis class.
Code lines: 16
Contained by: file galacticus.output.analyses.multi.F90

subroutine: `multifinalize`
Description: Finalize all analyses.

Code lines: 12
Contained by: file galacticus.output.analyses.multi.F90

function: multiloglikelihood
Description: Find the log-likelihood over all analyses. This asumes that the analyses are independent.
Code lines: 13
Contained by: file galacticus.output.analyses.multi.F90

subroutine: multireduce
Description: Reduce over the analysis.
Code lines: 21
Contained by: file galacticus.output.analyses.multi.F90
Modules used: galacticus_error

interface: outputanalysismulti
Description: Constructors for the multi merger tree analysis.
Code lines: 4
Contained by: file galacticus.output.analyses.multi.F90

file: galacticus.output.analyses.null.F90
Description: Contains a module which implements a null output analysis class.
Code lines: 92

subroutine: nullanalyze
Description: Implement a null output analysis.
Code lines: 9
Contained by: file galacticus.output.analyses.null.F90

function: nullconstructorparameters
Description: Constructor for the "null" output analysis class which takes a parameter set as input.
Code lines: 10
Contained by: file galacticus.output.analyses.null.F90
Modules used: input_parameters

subroutine: nullfinalize
Description: Implement a null output analysis finalization.
Code lines: 7
Contained by: file galacticus.output.analyses.null.F90

function: nullloglikelihood
Description: Return the log-likelihood of a null output analysis.
Code lines: 8
Contained by: file galacticus.output.analyses.null.F90

subroutine: nullreduce
Description: Implement a null output analysis reduction.
Code lines: 8
Contained by: file galacticus.output.analyses.null.F90

interface: `outputanalysisnull`
Description: Constructors for the "null" output analysis class.
Code lines: 3
Contained by: file galacticus.output.analyses.null.F90

file: `galacticus.output.analyses.options.F90`
Description: Contains a module which provides options and enuemrations for on-the-fly analyses.
Code lines: 51

module: `output_analyses_options`
Description: Provides options and enuemrations for on-the-fly analyses.
Code lines: 29
Contained by: file galacticus.output.analyses.options.F90
Used by:

function colordistributionsdssconstructorinternal	function concentrationdistributioncdmcococonstructorinter
function grvtnllnsngoperatedistribution	function magnificationcdfintegrand
function spinnbodyerrorsoperatescalar	function galaxysizessdssconstructorinternal
function luminosityfunctionconstructorinternal	function luminosityfunctionhalphaconstructorinternal
function massfunctionhiconstructorinternal	function massfunctionstellarconstructorinternal
file galacticus.output.analyses.mean_function_1d.F90	function antilog10operate
function csmlgyluminositydistanceoperate	function log10operate
function magnitudeoperate	file galacticus.output.analyses.scatter_function_1d.F90
function spindistributionbett2007constructorinternal	file galacticus.output.analyses.volume_function_1d.F90
function csmlgyvolumeoperate	module node_property_extractors
function icmsztype	function icmxrayluminositytype
function rateinfallcoldmodetype	function concentrationtype
function radiuscoolingtype	function ratecoolingtype
function densityprofiletype	function densitycontraststype
function descendentstype	function finaldescendenttype
function radiihalflightpropertiestype	function radiushalfmasstype
function halobiastype	function haloenvironmenttype
function masshosttype	function indiceshosttype
function fractionaccretionhotmodetype	function lightconetype
function lmnstyemssnlinequantity	function lmnstyemssnlinetype
function luminositystellarquantity	function luminositystellartype
function lmnstystllrchrltfll2000quantity	function lmnstystllrchrltfll2000type

function mainbranchstatustype
function massismtype
function massblackholetype
function massprofiletype
function massstellartype
function massstellarspheroidquantity
function metallicityismtype
function multitype
function nulltype
function halfmassradiustype
function redshiftlastisolatedtype
function satelliteorbitalextrematype
function spinbullocktype
function indicestreetype
function velocitydispersiontype
function virialpropertiestype
function massismquantity
function massblackholequantity
function masshalotype
function massstellarquantity
function massstellarmorphologytype
function massstellarspheroidtype
function mostmassiveprogenitortype
function nodeindicestype
function projecteddensitytype
function ratiotype
function rotationcurvetype
function satellitestatustype
function spintype
function treeweighttype
function velocitymaximumtype

file: galacticus.output.analyses.property_operator.F90
Description: Contains a module which provides a class that implements on-the-fly analyses.
Code lines: 44

module: output_analysis_property_operators
Description: Provides a class that implements operators on properties for on-the-fly analyses.
Code lines: 22
Contained by: file galacticus.output.analyses.property_operator.F90
Modules used: galacticus_nodes
iso_c_binding
Used by: subroutine galacticus_function_classes_destroy
function hivshalomassrelationpadmanabhan2017constructorin
function localgroupmassfunctionconstructorinternal
function blackholebulgerelationconstructorinternal
function concentrationvshalomasscdmludlow2016constructorinternal
file galacticus.output.analyses.correlation_function.F90
function correlationfunctionhearin2013sdssconstructorinternal
function massmetallicityandrews2013constructorinternal
function massmetallicityblanc2017constructorinternal
file galacticus.output.analyses.mean_function_1d.F90
function morphologicalfractiongamamoffett2016constructorinternal
file galacticus.output.analyses.scatter_function_1d.F90
function stellarvshalomassrelationleauthaud2012constructorinternal
file galacticus.output.analyses.volume_function_1d.F90
file galacticus.output.analyses.weight_operator.normal.F90
file galacticus.output.analyses.weight_operator.property.F90

subroutine galacticus_state_retrieve subroutine galacticus_state_store

file: galacticus.output.analyses.property_operator.HI_mass.F90
Description: Contains a module which implements a conversion of ISM mass to HI mass analysis property operator class.
Code lines: 97
Modules used: output_analysis_molecular_ratios

function: himassconstructorinternal
Description: Internal constructor for the "hiMass" output analysis distribution operator class.
Code lines: 9
Contained by: file galacticus.output.analyses.property_operator.HI_mass.F90
Modules used: input_parameters

function: himassconstructorparameters
Description: Constructor for the "hiMass" output analysis property operator class which takes a parameter set as input.
Code lines: 14
Contained by: file galacticus.output.analyses.property_operator.HI_mass.F90
Modules used: input_parameters

subroutine: himassdestructor
Description: Destructor for the "hiMass" output analysis distribution operator class.
Code lines: 8
Contained by: file galacticus.output.analyses.property_operator.HI_mass.F90
Modules used: input_parameters

function: himassoperate
Description: Implement an hiMass output analysis property operator.
Code lines: 16
Contained by: file galacticus.output.analyses.property_operator.HI_mass.F90
Modules used: galacticus_error iso_c_binding numerical_constants_astronomical

interface: outputanalysispropertyoperatorhimass
Description: Constructors for the "hiMass" output analysis class.
Code lines: 4
Contained by: file galacticus.output.analyses.property_operator.HI_mass.F90

file: galacticus.output.analyses.property_operator.antiLog10.F90
Description: Contains a module which implements an anti-$\log_{10}()$ output analysis property operator class.
Code lines: 73

function: antilog10constructorparameters
Description: Constructor for the "antiLog10" output analysis property operator class which takes a parameter set as input.
Code lines: 10
Contained by: file galacticus.output.analyses.property_operator.antiLog10.F90
Modules used: input_parameters

function: `antilog10operate`
Description: Implement an antiLog10 output analysis property operator.
Code lines: 22
Contained by: file galacticus.output.analyses.property_operator.antiLog10.F90
Modules used: iso_c_binding output_analyses_options

interface: `outputanalysispropertyoperatorantilog10`
Description: Constructors for the "antiLog10" output analysis class.
Code lines: 3
Contained by: file galacticus.output.analyses.property_operator.antiLog10.F90

file: `galacticus.output.analyses.property_operator.boolean.F90`
Description: Contains a module which implements a boolean analysis property operator class.
Code lines: 68

function: `booleanconstructorparameters`
Description: Constructor for the "boolean" output analysis property operator class which takes a parameter set as input.
Code lines: 10
Contained by: file galacticus.output.analyses.property_operator.boolean.F90
Modules used: input_parameters

function: `booleanoperate`
Description: Implement an boolean output analysis property operator.
Code lines: 17
Contained by: file galacticus.output.analyses.property_operator.boolean.F90
Modules used: iso_c_binding

interface: `outputanalysispropertyoperatorboolean`
Description: Constructors for the "boolean" output analysis class.
Code lines: 3
Contained by: file galacticus.output.analyses.property_operator.boolean.F90

file: `galacticus.output.analyses.property_operator.cosmology.angular_distance.F90`
Description: Contains a module which implements a cosmological angular distance corrector analysis property operator class.
Code lines: 140
Modules used: cosmology_functions iso_c_binding output_times

function: `csmlgyangulardistanceconstructorinternal`
Description: Internal constructor for the "randomErrorPolynomial" output analysis property operator class.
Code lines: 37
Contained by: file galacticus.output.analyses.property_operator.cosmology.angular_distance.F90
Modules used: galacticus_error iso_c_binding memory_management

function: csmlgyangulardistanceconstructorparameters
Description: Constructor for the "csmlgyAngularDistance" output analysis property operator class which takes a parameter set as input.
Code lines: 22
Contained by: file galacticus.output.analyses.property_operator.cosmology.angular_distance.F90
Modules used: input_parameters

subroutine: csmlgyangulardistancedestructor
Description: Destructor for the "csmlgyAnglrDstnc" output analysis property operator class.
Code lines: 11
Contained by: file galacticus.output.analyses.property_operator.cosmology.angular_distance.F90
Modules used: memory_management

function: csmlgyangulardistanceoperate
Description: Implement an csmlgyAngularDistance output analysis property operator.
Code lines: 16
Contained by: file galacticus.output.analyses.property_operator.cosmology.angular_distance.F90
Modules used: galacticus_error

interface: outputanalysispropertyoperatorcsmlgyanglrdstnc
Description: Constructors for the "csmlgyAngularDistance" output analysis class.
Code lines: 4
Contained by: file galacticus.output.analyses.property_operator.cosmology.angular_distance.F90

file: galacticus.output.analyses.property_operator.cosmology.luminosity_distance.F90
Description: Contains a module which implements a cosmological luminosity distance corrector analysis property operator class.
Code lines: 150
Modules used: cosmology_functions iso_c_binding output_times

function: csmlgyluminositydistanceconstructorinternal
Description: Internal constructor for the "randomErrorPolynomial" output analysis property operator class.
Code lines: 38
Contained by: file galacticus.output.analyses.property_operator.cosmology.luminosity_distance.F90
Modules used: galacticus_error iso_c_binding memory_management

function: csmlgyluminositydistanceconstructorparameters
Description: Constructor for the "csmlgyLuminosityDistance" output analysis property operator class which takes a parameter set as input.
Code lines: 22
Contained by: file galacticus.output.analyses.property_operator.cosmology.luminosity_distance.F90
Modules used: input_parameters

subroutine: csmlgyluminositydistancedestructor
Description: Destructorfor the "randomErrorPolynomial" output analysis property operator class.

Code lines: 11
Contained by: file galacticus.output.analyses.property_operator.cosmology.luminosity_distance.F90
Modules used: memory_management

function: csmlgyluminositydistanceoperate
Description: Implement an csmlgyLuminosityDistance output analysis property operator.
Code lines: 25
Contained by: file galacticus.output.analyses.property_operator.cosmology.luminosity_distance.F90
Modules used: galacticus_error output_analyses_options

interface: outputanalysispropertyoperatorcsmlgylmnstydstnc
Description: Constructors for the "csmlgyLuminosityDistance" output analysis class.
Code lines: 4
Contained by: file galacticus.output.analyses.property_operator.cosmology.luminosity_distance.F90

file: galacticus.output.analyses.property_operator.filter.high_pass.F90
Description: Contains a module which implements a high-pass filter analysis property operator class.
Code lines: 91

function: filterhighpassconstructorinternal
Description: Internal constructor for the "filterHighPass" output analysis distribution operator class.
Code lines: 9
Contained by: file galacticus.output.analyses.property_operator.filter.high_pass.F90
Modules used: input_parameters

function: filterhighpassconstructorparameters
Description: Constructor for the "filterHighPass" output analysis property operator class which takes a parameter set as input.
Code lines: 20
Contained by: file galacticus.output.analyses.property_operator.filter.high_pass.F90
Modules used: input_parameters

function: filterhighpassoperate
Description: Implement an filterHighPass output analysis property operator.
Code lines: 17
Contained by: file galacticus.output.analyses.property_operator.filter.high_pass.F90
Modules used: iso_c_binding

interface: outputanalysispropertyoperatorfilterhighpass
Description: Constructors for the "filterHighPass" output analysis class.
Code lines: 4
Contained by: file galacticus.output.analyses.property_operator.filter.high_pass.F90

file: galacticus.output.analyses.property_operator.identity.F90
Description: Contains a module which implements an identity output analysis property operator class.
Code lines: 64

function: identityconstructorparameters
Description: Constructor for the "identity" output analysis property operateor class which takes a parameter set as input.

Code lines: 10
Contained by: file galacticus.output.analyses.property_operator.identity.F90
Modules used: input_parameters

function: identityoperate
Description: Implement an identity output analysis property operator.
Code lines: 13
Contained by: file galacticus.output.analyses.property_operator.identity.F90
Modules used: iso_c_binding

interface: outputanalysispropertyoperatoridentity
Description: Constructors for the "identity" output analysis class.
Code lines: 3
Contained by: file galacticus.output.analyses.property_operator.identity.F90

file: galacticus.output.analyses.property_operator.log10.F90
Description: Contains a module which implements an log10 output analysis property operator class.
Code lines: 77

function: log10constructorparameters
Description: Constructor for the "log10" output analysis property operator class which takes a parameter set as input.
Code lines: 10
Contained by: file galacticus.output.analyses.property_operator.log10.F90
Modules used: input_parameters

function: log10operate
Description: Implement an log10 output analysis property operator.
Code lines: 26
Contained by: file galacticus.output.analyses.property_operator.log10.F90
Modules used: iso_c_binding output_analyses_options

interface: outputanalysispropertyoperatorlog10
Description: Constructors for the "log10" output analysis class.
Code lines: 3
Contained by: file galacticus.output.analyses.property_operator.log10.F90

file: galacticus.output.analyses.property_operator.magnitude.F90
Description: Contains a module which implements an output analysis property operator class which converts luminosity to absolute magnitude.
Code lines: 77

function: magnitudeconstructorparameters
Description: Constructor for the "magnitude" output analysis property operator class which takes a parameter set as input.
Code lines: 10
Contained by: file galacticus.output.analyses.property_operator.magnitude.F90
Modules used: input_parameters

function: magnitudeoperate

Description: Implement an magnitude output analysis property operator.
Code lines: 26
Contained by: file galacticus.output.analyses.property_operator.magnitude.F90
Modules used: iso_c_binding output_analyses_options

interface: outputanalysispropertyoperatormagnitude
Description: Constructors for the "magnitude" output analysis class.
Code lines: 3
Contained by: file galacticus.output.analyses.property_operator.magnitude.F90

file: galacticus.output.analyses.property_operator.metallicity.F90
Description: Contains a module which implements a property operator class which converts a metallicity, assumed to be a mass ratio of a given element to hydrogen, to $12 + \log_{10}(\mathrm{N/H})$ form.
Code lines: 95

function: metallicity12lognhconstructorinternal
Description: Internal constructor for the "metallicity12LogNH" output analysis distribution operator class.
Code lines: 8
Contained by: file galacticus.output.analyses.property_operator.metallicity.F90

function: metallicity12lognhconstructorparameters
Description: Constructor for the "metallicity12LogNH" output analysis property operator class which takes a parameter set as input.
Code lines: 20
Contained by: file galacticus.output.analyses.property_operator.metallicity.F90
Modules used: abundances_structure input_parameters

function: metallicity12lognhoperate
Description: Implement an metallicity output analysis property operator.
Code lines: 20
Contained by: file galacticus.output.analyses.property_operator.metallicity.F90
Modules used: iso_c_binding numerical_constants_atomic

interface: outputanalysispropertyoperatormetallicity12lognh
Description: Constructors for the "metallicity" output analysis class.
Code lines: 4
Contained by: file galacticus.output.analyses.property_operator.metallicity.F90

file: galacticus.output.analyses.property_operator.minmax.F90
Description: Contains a module which implements a min-max analysis property operator class.
Code lines: 93

function: minmaxconstructorinternal
Description: Internal constructor for the "minMax" output analysis distribution operator class.
Code lines: 9
Contained by: file galacticus.output.analyses.property_operator.minmax.F90
Modules used: input_parameters

function: `minmaxconstructorparameters`
Description: Constructor for the "minMax" output analysis property operator class which takes a parameter set as input.
Code lines: 26
Contained by: file `galacticus.output.analyses.property_operator.minmax.F90`
Modules used: `input_parameters`

function: `minmaxoperate`
Description: Implement an minMax output analysis property operator.
Code lines: 13
Contained by: file `galacticus.output.analyses.property_operator.minmax.F90`
Modules used: `iso_c_binding`

interface: `outputanalysispropertyoperatorminmax`
Description: Constructors for the "minMax" output analysis class.
Code lines: 4
Contained by: file `galacticus.output.analyses.property_operator.minmax.F90`

file: `galacticus.output.analyses.property_operator.multiply.F90`
Description: Contains a module which implements a multiplication analysis property operator class.
Code lines: 87

function: `multiplyconstructorinternal`
Description: Internal constructor for the "multiply" output analysis distribution operator class.
Code lines: 9
Contained by: file `galacticus.output.analyses.property_operator.multiply.F90`
Modules used: `input_parameters`

function: `multiplyconstructorparameters`
Description: Constructor for the "multiply" output analysis property operator class which takes a parameter set as input.
Code lines: 20
Contained by: file `galacticus.output.analyses.property_operator.multiply.F90`
Modules used: `input_parameters`

function: `multiplyoperate`
Description: Implement an multiply output analysis property operator.
Code lines: 13
Contained by: file `galacticus.output.analyses.property_operator.multiply.F90`
Modules used: `iso_c_binding`

interface: `outputanalysispropertyoperatormultiply`
Description: Constructors for the "multiply" output analysis class.
Code lines: 4
Contained by: file `galacticus.output.analyses.property_operator.multiply.F90`

file: `galacticus.output.analyses.property_operator.normal.F90`
Description: Contains a module which implements a property operator class in which the property value is replaced with an integral over a normal distribution between given limits, using the property value at the mean of the distribution.

Code lines: 118

function: `normalconstructorinternal`
Description: Internal constructor for the "normal" output analysis distribution operator class.
Code lines: 9
Contained by: file `galacticus.output.analyses.property_operator.normal.F90`
Modules used: `input_parameters`

function: `normalconstructorparameters`
Description: Constructor for the "normal" output analysis property operator class which takes a parameter set as input.
Code lines: 49
Contained by: file `galacticus.output.analyses.property_operator.normal.F90`
Modules used: `input_parameters`

function: `normaloperate`
Description: Implement an normal output analysis property operator.
Code lines: 14
Contained by: file `galacticus.output.analyses.property_operator.normal.F90`
Modules used: `error_functions` `iso_c_binding`

interface: `outputanalysispropertyoperatornormal`
Description: Constructors for the "normal" output analysis class.
Code lines: 4
Contained by: file `galacticus.output.analyses.property_operator.normal.F90`

file: `galacticus.output.analyses.property_operator.sequence.F90`
Description: Contains a module which implements a sequence output analysis property operator class.
Code lines: 181

interface: `outputanalysispropertyoperatorsequence`
Description: Constructors for the "sequence" output analysis class.
Code lines: 4
Contained by: file `galacticus.output.analyses.property_operator.sequence.F90`

type: `propertyoperatorlist`
Code lines: 3
Contained by: file `galacticus.output.analyses.property_operator.sequence.F90`

function: `sequenceconstructorinternal`
Description: Internal constructor for the sequence merger tree normalizer class.
Code lines: 14
Contained by: file `galacticus.output.analyses.property_operator.sequence.F90`

function: `sequenceconstructorparameters`
Description: Constructor for the "sequence" output analysis property operator class which takes a parameter set as input.
Code lines: 22
Contained by: file `galacticus.output.analyses.property_operator.sequence.F90`
Modules used: `input_parameters`

subroutine: `sequencedeepcopy`
Description: Perform a deep copy for the `sequence` output analysis property operator class.
Code lines: 31
Contained by: file `galacticus.output.analyses.property_operator.sequence.F90`
Modules used: `galacticus_error`

subroutine: `sequencedestructor`
Description: Destructor for the sequence property operator class.
Code lines: 16
Contained by: file `galacticus.output.analyses.property_operator.sequence.F90`

function: `sequenceoperate`
Description: Implement an sequence output analysis property operator.
Code lines: 17
Contained by: file `galacticus.output.analyses.property_operator.sequence.F90`

subroutine: `sequenceprepend`
Description: Prepend an operator to the sequence.
Code lines: 12
Contained by: file `galacticus.output.analyses.property_operator.sequence.F90`

file: `galacticus.output.analyses.property_operator.square.F90`
Description: Contains a module which implements a square output analysis property operator class.
Code lines: 65

interface: `outputanalysispropertyoperatorsquare`
Description: Constructors for the "square" output analysis class.
Code lines: 3
Contained by: file `galacticus.output.analyses.property_operator.square.F90`

function: `squareconstructorparameters`
Description: Constructor for the "square" output analysis property operateor class which takes a parameter set as input.
Code lines: 10
Contained by: file `galacticus.output.analyses.property_operator.square.F90`
Modules used: `input_parameters`

function: `squareoperate`
Description: Implement an square root output analysis property operator.
Code lines: 14
Contained by: file `galacticus.output.analyses.property_operator.square.F90`
Modules used: `galacticus_error` `iso_c_binding`

file: `galacticus.output.analyses.property_operator.square_root.F90`
Description: Contains a module which implements an square root output analysis property operator class.
Code lines: 66

interface: `outputanalysispropertyoperatorsquareroot`

Description: Constructors for the "squareRoot" output analysis class.
Code lines: 3
Contained by: file galacticus.output.analyses.property_operator.square_root.F90

function: squarerootconstructorparameters
Description: Constructor for the "squareRoot" output analysis property operateor class which takes a parameter set as input.
Code lines: 10
Contained by: file galacticus.output.analyses.property_operator.square_root.F90
Modules used: input_parameters

function: squarerootoperate
Description: Implement an square root output analysis property operator.
Code lines: 15
Contained by: file galacticus.output.analyses.property_operator.square_root.F90
Modules used: galacticus_error iso_c_binding

file: galacticus.output.analyses.property_operator.systematic.polynomial.F90
Description: Contains a module which implements a polynomial systematic shift output analysis property operator class.
Code lines: 103

interface: outputanalysispropertyoperatorsystmtcpolynomial
Description: Constructors for the "systmtcPolynomial" output analysis class.
Code lines: 4
Contained by: file galacticus.output.analyses.property_operator.systematic.polynomial.F90

function: systmtcpolynomialconstructorinternal
Description: Internal constructor for the "randomErrorPolynomial" output analysis distribution operator class.
Code lines: 9
Contained by: file galacticus.output.analyses.property_operator.systematic.polynomial.F90

function: systmtcpolynomialconstructorparameters
Description: Constructor for the "systmtcPolynomial" output analysis property operator class which takes a parameter set as input.
Code lines: 31
Contained by: file galacticus.output.analyses.property_operator.systematic.polynomial.F90
Modules used: input_parameters

function: systmtcpolynomialoperate
Description: Implement an systmtcPolynomial output analysis property operator.
Code lines: 17
Contained by: file galacticus.output.analyses.property_operator.systematic.polynomial.F90
Modules used: iso_c_binding

file: galacticus.output.analyses.scatter_function_1d.F90
Description: Contains a module which implements a generic 1D scatter function (i.e. the scatter of some property weighted by number density of objects binned by some property) output analysis class.
Code lines: 651

Modules used: galactic_filters, iso_c_binding, iso_varying_string, node_property_extractors, output_analyses_options, output_analysis_distribution_-normalizers, output_analysis_distribution_-operators, output_analysis_property_operators, output_analysis_weight_operators, output_times

interface: `outputanalysisscatterfunction1d`
Description: Constructors for the "scatterFunction1D" output analysis class.
Code lines: 4
Contained by: file galacticus.output.analyses.scatter_function_1d.F90

subroutine: `scatterfunction1danalyze`
Description: Implement a scatterFunction1D output analysis.
Code lines: 11
Contained by: file galacticus.output.analyses.scatter_function_1d.F90

function: `scatterfunction1dconstructorinternal`
Description: Constructor for the "scatterFunction1D" output analysis class for internal use.
Code lines: 116
Contained by: file galacticus.output.analyses.scatter_function_1d.F90
Modules used: memory_management

function: `scatterfunction1dconstructorparameters`
Description: Constructor for the "scatterFunction1D" output analysis class which takes a parameter set as input.
Code lines: 300
Contained by: file galacticus.output.analyses.scatter_function_1d.F90
Modules used: galacticus_error, input_parameters, memory_management

subroutine: `scatterfunction1ddestructor`
Description: Destructor for the `scatterFunction1D` output analysis class.
Code lines: 8
Contained by: file galacticus.output.analyses.scatter_function_1d.F90

subroutine: `scatterfunction1dfinalize`
Description: Implement a `scatterFunction1D` output analysis finalization.
Code lines: 56
Contained by: file galacticus.output.analyses.scatter_function_1d.F90
Modules used: galacticus_hdf5, io_hdf5

subroutine: `scatterfunction1dfinalizeanalysis`
Description: Finalize analysis of a `scatterFunction1D` output analysis.
Code lines: 32
Contained by: file galacticus.output.analyses.scatter_function_1d.F90

function: `scatterfunction1dloglikelihood`

Description: Return the log-likelihood of a scatterFunction1D output analysis.
Code lines: 32
Contained by: file galacticus.output.analyses.scatter_function_1d.F90
Modules used: galacticus_error linear_algebra numerical_constants_math

subroutine: scatterfunction1dreduce
Description: Implement a scatterFunction1D output analysis reduction.
Code lines: 15
Contained by: file galacticus.output.analyses.scatter_function_1d.F90
Modules used: galacticus_error

file: galacticus.output.analyses.spin_distribution.Bett2007.F90
Description: Contains a module which implements a spin parameter distribution output analysis class.
Code lines: 271
Modules used: cosmology_functions dark_matter_halo_scales dark_matter_profile_scales dark_matter_profiles_dmo halo_mass_functions statistics_nbody_halo_mass_errors

interface: outputanalysisspindistributionbett2007
Description: Constructors for the "spinDistributionBett2007" output analysis class.
Code lines: 4
Contained by: file galacticus.output.analyses.spin_distribution.Bett2007.F90

function: spindistributionbett2007constructorinternal
Description: Internal constructor for the "spinDistributionBett2007" output analysis class.
Code lines: 182
Contained by: file galacticus.output.analyses.spin_distribution.Bett2007.F90
Modules used: galacticus_error galacticus_paths halo_spin_distributions io_hdf5 iso_varying_string memory_management numerical_comparison output_analyses_options output_times virial_density_contrast

function: spindistributionbett2007constructorparameters
Description: Constructor for the "spinDistributionBett2007" output analysis class which takes a parameter set as input.
Code lines: 42
Contained by: file galacticus.output.analyses.spin_distribution.Bett2007.F90
Modules used: functions_global input_parameters

file: galacticus.output.analyses.stellar_vs_halo_mass_relation.COSMOS_Leauthaud2012.F90
Description: Contains a module which implements a stellar vs halo mass relation analysis class.
Code lines: 508
Modules used: iso_c_binding

interface: outputanalysisstellarvshalomassrelationleauthaud2012
Description: Constructors for the "stellarVsHaloMassRelationLeauthaud2012" output analysis class.
Code lines: 4

Contained by: file galacticus.output.analyses.stellar_vs_halo_mass_relation.COSMOS_Leauthaud2012.F90

subroutine: stellarvshalomassrelationleauthaud2012analyze
Description: Implement a stellarVsHaloMassRelationLeauthaud2012 output analysis.
Code lines: 9
Contained by: file galacticus.output.analyses.stellar_vs_halo_mass_relation.COSMOS_Leauthaud2012.F90

function: stellarvshalomassrelationleauthaud2012constructorinternal
Description: Constructor for the "stellarVsHaloMassRelationLeauthaud2012" output analysis class for internal use.
Code lines: 338
Contained by: file galacticus.output.analyses.stellar_vs_halo_mass_relation.COSMOS_Leauthaud2012.F90
Modules used:

cosmology_functions	cosmology_parameters
fgsl	galacticus_error
galacticus_paths	io_hdf5
iso_varying_string	memory_management
node_property_extractors	numerical_comparison
numerical_constants_astronomical	numerical_ranges
output_analysis_distribution_operators	output_analysis_property_operators
output_analysis_utilities	output_analysis_weight_operators
string_handling	tables
virial_density_contrast	

function: stellarvshalomassrelationleauthaud2012constructorparameters
Description: Constructor for the "stellarVsHaloMassRelationLeauthaud2012" output analysis class which takes a parameter set as input.
Code lines: 66
Contained by: file galacticus.output.analyses.stellar_vs_halo_mass_relation.COSMOS_Leauthaud2012.F90
Modules used:

cosmology_functions	cosmology_parameters
input_parameters	

subroutine: stellarvshalomassrelationleauthaud2012destructor
Description: Destructor for the stellarVsHaloMassRelationLeauthaud2012 output analysis class.
Code lines: 7
Contained by: file galacticus.output.analyses.stellar_vs_halo_mass_relation.COSMOS_Leauthaud2012.F90

subroutine: stellarvshalomassrelationleauthaud2012finalize
Description: Implement a stellarVsHaloMassRelationLeauthaud2012 output analysis finalization.
Code lines: 7
Contained by: file galacticus.output.analyses.stellar_vs_halo_mass_relation.COSMOS_Leauthaud2012.F90

function: stellarvshalomassrelationleauthaud2012loglikelihood
Description: Return the log-likelihood of a stellarVsHaloMassRelationLeauthaud2012 output analysis.
Code lines: 7
Contained by: file galacticus.output.analyses.stellar_vs_halo_mass_relation.COSMOS_Leauthaud2012.F90

subroutine: stellarvshalomassrelationleauthaud2012reduce
Description: Implement reduction for the stellarVsHaloMassRelationLeauthaud2012 output analysis class.
Code lines: 14

Contained by: file galacticus.output.analyses.stellar_vs_halo_mass_relation.COSMOS_Leauthaud2012.F90
Modules used: galacticus_error

file: galacticus.output.analyses.utilities.F90
Description: Contains a module which provides a collection of utilities useful for on-the-fly analyses.
Code lines: 122

module: output_analysis_utilities
Description: Provides a collection of utilities useful for on-the-fly analyses.
Code lines: 100
Contained by: file galacticus.output.analyses.utilities.F90
Used by:

function	function
hivshalomassrelationpadmanabhan2017constructorinternal	blackholemassrelationconstructorinternal
function	function
colordistributionsdssconstructorinternal	concentrationdistributioncdmcococonstructorinter
function	function
concentrationvshalomasscdmludlow2016constructorinternal	correlationfunctionconstructorinternal
function	function
galaxysizessdssconstructorinternal	luminosityfunctionconstructorinternal
function	function
luminosityfunctionhalphaconstructorinternal	massfunctionhiconstructorinternal
function	function
massfunctionstellarconstructorinternal	massmetallicityandrews2013constructorinternal
function	function
massmetallicityblanc2017constructorinternal	morphologicalfractiongamamoffett2016constructori
function	
stellarvshalomassrelationleauthaud2012constructorinternal	

function: output_analysis_output_weight_survey_volume
Description: Compute output weights corresponding to the cosmological volumes associated with the given survey.
Code lines: 90
Contained by: module output_analysis_utilities
Modules used:

cosmology_functions	galacticus_error
geometry_surveys	iso_c_binding
iso_varying_string	output_times

file: galacticus.output.analyses.volume_function_1d.F90
Description: Contains a module which implements a generic 1D volume function (i.e. number density of objects binned by some property, e.g. a mass function) output analysis class.
Code lines: 751
Modules used:

galactic_filters	iso_c_binding
iso_varying_string	node_property_extractors
output_analyses_options	output_analysis_distribution_-normalizers
output_analysis_distribution_-operators	output_analysis_property_operators
output_analysis_weight_operators	output_times

interface: `outputanalysisvolumefunction1d`
Description: Constructors for the "volumeFunction1D" output analysis class.
Code lines: 4
Contained by: file `galacticus.output.analyses.volume_function_1d.F90`

subroutine: `volumefunction1danalyze`
Description: Implement a volumeFunction1D output analysis.
Code lines: 73
Contained by: file `galacticus.output.analyses.volume_function_1d.F90`
Modules used: `galacticus_nodes`

function: `volumefunction1dconstructorinternal`
Description: Constructor for the "volumeFunction1D" output analysis class for internal use.
Code lines: 82
Contained by: file `galacticus.output.analyses.volume_function_1d.F90`
Modules used: `galacticus_error` `memory_management`

function: `volumefunction1dconstructorparameters`
Description: Constructor for the "volumeFunction1D" output analysis class which takes a parameter set as input.
Code lines: 297
Contained by: file `galacticus.output.analyses.volume_function_1d.F90`
Modules used: `galacticus_error` `input_parameters` `memory_management`

subroutine: `volumefunction1ddestructor`
Description: Destructor for the "volumeFunction1D" output analysis class.
Code lines: 15
Contained by: file `galacticus.output.analyses.volume_function_1d.F90`

subroutine: `volumefunction1dfinalize`
Description: Implement a volumeFunction1D output analysis finalization.
Code lines: 56
Contained by: file `galacticus.output.analyses.volume_function_1d.F90`
Modules used: `galacticus_hdf5` `io_hdf5`

subroutine: `volumefunction1dfinalizeanalysis`
Description: Compute final covariances and normalize.
Code lines: 47
Contained by: file `galacticus.output.analyses.volume_function_1d.F90`
Modules used: `mpi_utilities`

function: `volumefunction1dloglikelihood`
Description: Return the log-likelihood of a volumeFunction1D output analysis.
Code lines: 32
Contained by: file `galacticus.output.analyses.volume_function_1d.F90`
Modules used: `galacticus_error` `linear_algebra` `numerical_constants_math`

subroutine: `volumefunction1dreduce`
Description: Implement a volumeFunction1D output analysis reduction.
Code lines: 21
Contained by: file galacticus.output.analyses.volume_function_1d.F90
Modules used: galacticus_error

subroutine: `volumefunction1dresults`
Description: Implement a volumeFunction1D output analysis finalization.
Code lines: 27
Contained by: file galacticus.output.analyses.volume_function_1d.F90
Modules used: memory_management

file: `galacticus.output.analyses.weight_operator.F90`
Description: Contains a module which provides a class that implements weight on operators for on-the-fly analyses.
Code lines: 45

module: `output_analysis_weight_operators`
Description: Provides a class that implements operators on weights for on-the-fly analyses.
Code lines: 23
Contained by: file galacticus.output.analyses.weight_operator.F90
Modules used: galacticus_nodes, iso_c_binding
Used by: subroutine galacticus_function_classes_destroy, function hivshalomassrelationpadmanabhan2017constructorin, function localgroupmassfunctionconstructorinterna, function blackholebulgerelationconstructorinternal, function concentrationvshalomasscdmludlow2016constructorinternal, function massmetallicityandrews2013constructorinternal, function massmetallicityblanc2017constructorinternal, file galacticus.output.analyses.mean_function_1d.F90, function morphologicalfractiongamamoffett2016constructorinternal, file galacticus.output.analyses.scatter_function_1d.F90, function stellarvshalomassrelationleauthaud2012constructorinternal, file galacticus.output.analyses.volume_function_1d.F90, subroutine galacticus_state_retrieve, subroutine galacticus_state_store

file: `galacticus.output.analyses.weight_operator.N-body_mass.F90`
Description: Contains a module which implements a weight operator class in which the weight is multiplied by an integral over the N-body halo mass distribution.
Code lines: 135
Modules used: statistics_nbody_halo_mass_errors

function: `nbodymassconstructorinternal`
Description: Internal constructor for the "nbodyMass" output analysis distribution operator class.
Code lines: 18
Contained by: file galacticus.output.analyses.weight_operator.N-body_mass.F90

Modules used: galacticus_error

function: nbodymassconstructorparameters
Description: Constructor for the "nbodyMass" output analysis weight operator class which takes a parameter set as input.
Code lines: 35
Contained by: file galacticus.output.analyses.weight_operator.N-body_mass.F90
Modules used: input_parameters

subroutine: nbodymassdestructor
Description: Destructor for the "nbodyMass" output analysis weight operator class.
Code lines: 8
Contained by: file galacticus.output.analyses.weight_operator.N-body_mass.F90

function: nbodymassrootvariance
Description: Return the root variance for use in the "nbodyMass" output analysis weight operator class.
Code lines: 22
Contained by: file galacticus.output.analyses.weight_operator.N-body_mass.F90

interface: outputanalysisweightoperatornbodymass
Description: Constructors for the "nbodyMass" output analysis class.
Code lines: 4
Contained by: file galacticus.output.analyses.weight_operator.N-body_mass.F90

file: galacticus.output.analyses.weight_operator.cosmology.volume.F90
Description: Contains a module which implements a cosmological volume corrector analysis weight operator class.
Code lines: 162
Modules used: cosmology_functions geometry_surveys

function: csmlgyvolumeconstructorinternal
Description: Internal constructor for the "csmlgyVolume" output analysis weight operator class.
Code lines: 9
Contained by: file galacticus.output.analyses.weight_operator.cosmology.volume.F90

function: csmlgyvolumeconstructorparameters
Description: Constructor for the "csmlgyVolume" output analysis weight operator class which takes a parameter set as input.
Code lines: 22
Contained by: file galacticus.output.analyses.weight_operator.cosmology.volume.F90
Modules used: input_parameters

subroutine: csmlgyvolumedestructor
Description: Destructor for the "csmlgyVolume" output analysis weight operator class.
Code lines: 9
Contained by: file galacticus.output.analyses.weight_operator.cosmology.volume.F90

function: csmlgyvolumeoperate
Description: Implement an csmlgyVolume output analysis weight operator.

Code lines: 70
Contained by: file galacticus.output.analyses.weight_operator.cosmology.volume.F90
Modules used: galacticus_error iso_c_binding output_analyses_options output_times

interface: outputanalysisweightoperatorcsmlgyvolume
Description: Constructors for the "csmlgyVolume" output analysis class.
Code lines: 4
Contained by: file galacticus.output.analyses.weight_operator.cosmology.volume.F90

file: galacticus.output.analyses.weight_operator.filter.high_pass.F90
Description: Contains a module which implements a high-pass filter analysis weight operator class.
Code lines: 91

function: filterhighpassconstructorinternal
Description: Internal constructor for the "filterHighPass" output analysis distribution operator class.
Code lines: 9
Contained by: file galacticus.output.analyses.weight_operator.filter.high_pass.F90
Modules used: input_parameters

function: filterhighpassconstructorparameters
Description: Constructor for the "filterHighPass" output analysis weight operator class which takes a parameter set as input.
Code lines: 20
Contained by: file galacticus.output.analyses.weight_operator.filter.high_pass.F90
Modules used: input_parameters

function: filterhighpassoperate
Description: Implement an filterHighPass output analysis weight operator.
Code lines: 17
Contained by: file galacticus.output.analyses.weight_operator.filter.high_pass.F90
Modules used: iso_c_binding

interface: outputanalysisweightoperatorfilterhighpass
Description: Constructors for the "filterHighPass" output analysis class.
Code lines: 4
Contained by: file galacticus.output.analyses.weight_operator.filter.high_pass.F90

file: galacticus.output.analyses.weight_operator.identity.F90
Description: Contains a module which implements an identity analysis weight operator class.
Code lines: 65

function: identityconstructorparameters
Description: Constructor for the "identity" output analysis weight operator class which takes a parameter set as input.
Code lines: 11
Contained by: file galacticus.output.analyses.weight_operator.identity.F90
Modules used: input_parameters

function: `identityoperate`
Description: Implement an identity output analysis weight operator.
Code lines: 13
Contained by: file `galacticus.output.analyses.weight_operator.identity.F90`
Modules used: `iso_c_binding`

interface: `outputanalysisweightoperatoridentity`
Description: Constructors for the "identity" output analysis class.
Code lines: 3
Contained by: file `galacticus.output.analyses.weight_operator.identity.F90`

file: `galacticus.output.analyses.weight_operator.normal.F90`
Description: Contains a module which implements a weight operator class in which the weight is multiplied by an integral over a normal distribution.
Code lines: 173
Modules used: `node_property_extractors` `output_analysis_property_operators`

function: `normalconstructorinternal`
Description: Internal constructor for the "normal" output analysis distribution operator class.
Code lines: 18
Contained by: file `galacticus.output.analyses.weight_operator.normal.F90`
Modules used: `galacticus_error` `input_parameters`

function: `normalconstructorparameters`
Description: Constructor for the "normal" output analysis weight operator class which takes a parameter set as input.
Code lines: 40
Contained by: file `galacticus.output.analyses.weight_operator.normal.F90`
Modules used: `input_parameters`

subroutine: `normaldestructor`
Description: Destructor for the "normal" output analysis weight operator class.
Code lines: 7
Contained by: file `galacticus.output.analyses.weight_operator.normal.F90`

function: `normaloperate`
Description: Implement an normal output analysis weight operator.
Code lines: 31
Contained by: file `galacticus.output.analyses.weight_operator.normal.F90`
Modules used: `error_functions` `iso_c_binding`

function: `normalrootvariance`
Description: Return the root variance for use in the "normal" output analysis weight operator class.
Code lines: 12
Contained by: file `galacticus.output.analyses.weight_operator.normal.F90`

interface: `outputanalysisweightoperatornormal`
Description: Constructors for the "normal" output analysis class.

Code lines: 4
Contained by: file galacticus.output.analyses.weight_operator.normal.F90

file: galacticus.output.analyses.weight_operator.property.F90
Description: Contains a module which implements an analysis weight operator class which weights by a property value.
Code lines: 115
Modules used: node_property_extractors output_analysis_property_operators

interface: outputanalysisweightoperatorproperty
Description: Constructors for the "property" output analysis class.
Code lines: 4
Contained by: file galacticus.output.analyses.weight_operator.property.F90

function: propertyconstructorinternal
Description: Internal constructor for the "property" output analysis weight operator class.
Code lines: 16
Contained by: file galacticus.output.analyses.weight_operator.property.F90
Modules used: galacticus_error

function: propertyconstructorparameters
Description: Constructor for the "property" output analysis weight operator class which takes a parameter set as input.
Code lines: 18
Contained by: file galacticus.output.analyses.weight_operator.property.F90
Modules used: input_parameters

subroutine: propertydestructor
Description: Destructor for the "property" output analysis weight operator class.
Code lines: 8
Contained by: file galacticus.output.analyses.weight_operator.property.F90

function: propertyoperate
Description: Implement an property output analysis weight operator.
Code lines: 22
Contained by: file galacticus.output.analyses.weight_operator.property.F90
Modules used: iso_c_binding

file: galacticus.output.analyses.weight_operator.sequence.F90
Description: Contains a module which implements a sequence output analysis weight operator class.
Code lines: 180

interface: outputanalysisweightoperatorsequence
Description: Constructors for the "sequence" output analysis class.
Code lines: 4
Contained by: file galacticus.output.analyses.weight_operator.sequence.F90

function: sequenceconstructorinternal
Description: Internal constructor for the sequence output analysis weight operator class.

Code lines: 14
Contained by: file galacticus.output.analyses.weight_operator.sequence.F90

function: sequenceconstructorparameters
Description: Constructor for the "sequence" output analysis weight operator class which takes a parameter set as input.
Code lines: 22
Contained by: file galacticus.output.analyses.weight_operator.sequence.F90
Modules used: input_parameters

subroutine: sequencedeepcopy
Description: Perform a deep copy for the sequence output analysis weight operator class.
Code lines: 31
Contained by: file galacticus.output.analyses.weight_operator.sequence.F90
Modules used: galacticus_error

subroutine: sequencedestructor
Description: Destructor for the sequence weight operator class.
Code lines: 16
Contained by: file galacticus.output.analyses.weight_operator.sequence.F90

function: sequenceoperate
Description: Implement an sequence output analysis weight operator.
Code lines: 17
Contained by: file galacticus.output.analyses.weight_operator.sequence.F90

subroutine: sequenceprepend
Description: Prepend an operator to the sequence.
Code lines: 12
Contained by: file galacticus.output.analyses.weight_operator.sequence.F90

type: weightoperatorlist
Code lines: 3
Contained by: file galacticus.output.analyses.weight_operator.sequence.F90

file: galacticus.output.build.F90
Description: Contains a module which implements writing of Galacticus build information to the Galacticus output file.
Code lines: 212

module: galacticus_build
Description: Implements writing of Galacticus build information to the Galacticus output file.
Code lines: 187
Contained by: file galacticus.output.build.F90
Modules used: iso_c_binding
Used by: subroutine galacticus_output_open_file subroutine reportperform module input_parameters

subroutine: galacticus_build_output

Description: Output build information to the main output file.
Code lines: 94
Contained by: module galacticus_build
Modules used: fgsl, file_utilities, fox_common, galacticus_error, galacticus_hdf5, galacticus_paths, iso_varying_string, string_handling

function: galacticus_build_string
Description: Returns a string describing the build environment of Galacticus.
Code lines: 69
Contained by: module galacticus_build
Modules used: fgsl, fox_common, galacticus_error, hdf5, iso_varying_string, string_handling

file: galacticus.output.merger_tree.halo_model.F90
Description: Contains a module which handles outputting of data required by the halo model of galaxy clustering to the Galacticus output file.
Code lines: 196

module: galacticus_output_halo_models
Description: Handles outputting of data required by the halo model of galaxy clustering to the Galacticus output file.
Code lines: 174
Contained by: file galacticus.output.merger_tree.halo_model.F90
Used by: subroutine evolveforestsperform

subroutine: galacticus_extra_output_halo_fourier_profile
Description: Store Fourier-space halo profiles to the output file.
Code lines: 76
Contained by: module galacticus_output_halo_models
Modules used: cosmology_functions, dark_matter_profiles_dmo, galacticus_hdf5, galacticus_nodes, iso_c_binding, iso_varying_string, kind_numbers, memory_management, string_handling

subroutine: galacticus_output_halo_model_initialize
Description: Initializes the module by determining whether or not halo model data should be output.
Code lines: 69
Contained by: module galacticus_output_halo_models
Modules used: galacticus_hdf5, input_parameters, io_hdf5, numerical_constants_astronomical, numerical_ranges

file: galacticus.output.version.F90
Description: Contains a module which implements writing of the version number and run time to the Galacticus output file.
Code lines: 124

module: galacticus_versioning
Description: Implements writing of the version number and run time to the Galacticus output file.
Code lines: 102
Contained by: file galacticus.output.version.F90
Used by: subroutine galacticus_output_open_file, subroutine reportperform, module input_parameters

subroutine: galacticus_version
Description: Return version information
Code lines: 13
Contained by: module galacticus_versioning
Modules used: iso_varying_string

subroutine: galacticus_version_output
Description: Output version information to the main output file.
Code lines: 56
Contained by: module galacticus_versioning
Modules used: dates_and_times, file_utilities, fox_dom, fox_utils, galacticus_error, galacticus_hdf5, io_hdf5, io_xml, iso_varying_string

function: galacticus_version_string
Description: Returns a string describing the version of Galacticus.
Code lines: 9
Contained by: module galacticus_versioning
Modules used: iso_varying_string, string_handling

file: galacticus.state.F90
Description: Contains a module which implements storage and recovery of the Galacticus internal state. Used for restoring random number generator sequences for example.
Code lines: 887

module: galacticus_state
Description: Implements storage and recovery of the Galacticus internal state. Used for restoring random number generator sequences for example.
Code lines: 864
Contained by: file galacticus.state.F90
Modules used: iso_c_binding, iso_varying_string, tables
Used by: subroutine functions_global_set

subroutine: galacticus_state_retrieve
Description: Retrieve the internal state.
Code lines: 390
Contained by: module galacticus_state
Modules used: accretion_disk_spectra, accretion_disks

accretion_halo_totals
accretion_halos
atomic_cross_sections_ionization_photo
atomic_ionization_potentials
atomic_radiation_gaunt_factors
atomic_rates_excitation_collisional
atomic_rates_ionization_collisional
atomic_rates_recombination_dielectronic
atomic_rates_recombination_radiative
black_hole_binary_initial_separation
black_hole_binary_mergers
black_hole_binary_recoil_velocities
black_hole_binary_separations
chemical_reaction_rates
chemical_states
conditional_mass_functions
cooling_cold_mode_infall_rates
cooling_freefall_times_available
cooling_functions
cooling_infall_radii
cooling_radii
cooling_rates
cooling_specific_angular_momenta
cooling_times
cooling_times_available
cosmological_density_field
cosmology_functions
cosmology_parameters
dark_matter_halo_biases
dark_matter_halo_mass_accretion_histories
dark_matter_halo_scales
dark_matter_halos_mass_loss_rates
dark_matter_particles
dark_matter_profile_scales
dark_matter_profiles
dark_matter_profiles_concentration
dark_matter_profiles_dmo
dark_matter_profiles_shape
excursion_sets_barriers
excursion_sets_first_crossings
fgsl
freefall_radii
galactic_dynamics_bar_instabilities
galactic_filters
galactic_structure_solvers
geometry_lightcones
geometry_surveys
gravitational_lensing
halo_mass_functions
halo_model_power_spectrum_modifiers
halo_spin_distributions
hot_halo_cold_mode_density_core_radii
hot_halo_mass_distributions
hot_halo_mass_distributions_core_radii
hot_halo_outflows_reincorporations
hot_halo_ram_pressure_forces
hot_halo_ram_pressure_stripping
hot_halo_ram_pressure_stripping_timescales
hot_halo_temperature_profiles
intergalactic_medium_filtering_masses
intergalactic_medium_state
linear_growth
mass_distributions
mass_function_incompletenesses
math_operators_unary
merger_tree_branching
merger_tree_branching_modifiers
merger_tree_construction
merger_tree_operators
merger_tree_outputters
merger_tree_read_importers
merger_tree_state_store
merger_tree_timesteps
merger_tree_walkers
merger_trees_build_mass_resolution
merger_trees_build_masses
merger_trees_build_masses_distributions
merger_trees_builders

merger_trees_evolve
merger_trees_evolve_node
merger_trees_merge_node
meta_tree_compute_times
model_parameters
models_likelihoods
mpi_utilities
nbody_importers
nbody_operators
node_component_disk_standard
node_component_spheroid_standard
node_property_extractors
output_analyses
output_analysis_distribution_normalizers
output_analysis_distribution_operators
output_analysis_molecular_ratios
output_analysis_property_operators
output_analysis_weight_operators
output_times
posterior_sample_differential_proposal_size
posterior_sample_differential_random_jump
posterior_sampling_convergence
posterior_sampling_prop_size_temp_exp
posterior_sampling_simulation
posterior_sampling_state
posterior_sampling_state_initialize
posterior_sampling_stopping_criteria
power_spectra
power_spectra_nonlinear
power_spectra_primordial
power_spectra_primordial_transferred
power_spectrum_window_functions
radiation_fields
ram_pressure_stripping_mass_loss_rate_disks
ram_pressure_stripping_mass_loss_rate_spheroids
satellite_dynamical_friction
satellite_merging_mass_movements
satellite_merging_progenitor_properties
satellite_merging_remnant_sizes
satellite_merging_timescales
satellite_oprhan_distributions
satellite_tidal_heating
satellite_tidal_stripping
satellites_tidal_fields
star_formation_feedback_disks
star_formation_feedback_expulsion_disks
star_formation_feedback_expulsion_spheroids
star_formation_feedback_spheroids
star_formation_histories
star_formation_rate_surface_density_disks
star_formation_timescales_disks
star_formation_timescales_spheroids
statistics_nbody_halo_mass_errors
stellar_astrophysics
stellar_astrophysics_tracks
stellar_astrophysics_winds
stellar_feedback
stellar_luminosities_structure
stellar_population_properties
stellar_population_selectors
stellar_population_spectra
stellar_population_spectra_postprocess
stellar_populations
stellar_populations_initial_mass_functions
stellar_spectra_dust_attenuations
string_handling

supernovae_population_iii, supernovae_type_ia, task_evolve_forests_work_shares, tasks, tidal_stripping_mass_loss_rate_disks, tidal_stripping_mass_loss_rate_spheroids, transfer_functions, unevolved_subhalo_mass_functions, universe_operators, virial_density_contrast, virial_orbits

subroutine: galacticus_state_store

Description: Store the internal state.

Code lines: 395

Contained by: module galacticus_state

Modules used: accretion_disk_spectra, accretion_disks, accretion_halo_totals, accretion_halos, atomic_cross_sections_ionization_photo, atomic_ionization_potentials, atomic_radiation_gaunt_factors, atomic_rates_excitation_collisional, atomic_rates_ionization_collisional, atomic_rates_recombination_dielectronic, atomic_rates_recombination_radiative, black_hole_binary_initial_separation, black_hole_binary_mergers, black_hole_binary_recoil_velocities, black_hole_binary_separations, chemical_reaction_rates, chemical_states, conditional_mass_functions, cooling_cold_mode_infall_rates, cooling_freefall_times_available, cooling_functions, cooling_infall_radii, cooling_radii, cooling_rates, cooling_specific_angular_momenta, cooling_times, cooling_times_available, cosmological_density_field, cosmology_functions, cosmology_parameters, dark_matter_halo_biases, dark_matter_halo_mass_accretion_histories, dark_matter_halo_scales, dark_matter_halos_mass_loss_rates, dark_matter_particles, dark_matter_profile_scales, dark_matter_profiles, dark_matter_profiles_concentration, dark_matter_profiles_dmo, dark_matter_profiles_shape, excursion_sets_barriers, excursion_sets_first_crossings, fgsl, freefall_radii, galactic_dynamics_bar_instabilities, galactic_filters, galactic_structure_solvers, geometry_lightcones, geometry_surveys, gravitational_lensing, halo_mass_functions, halo_model_power_spectrum_modifiers, halo_spin_distributions, hot_halo_cold_mode_density_core_radii, hot_halo_mass_distributions, hot_halo_mass_distributions_core_radii, hot_halo_outflows_reincorporations, hot_halo_ram_pressure_forces, hot_halo_ram_pressure_stripping, hot_halo_ram_pressure_stripping_timescales

hot_halo_temperature_profiles
intergalactic_medium_filtering_masses
intergalactic_medium_state
linear_growth
mass_distributions
mass_function_incompletenesses
math_operators_unary
merger_tree_branching
merger_tree_branching_modifiers
merger_tree_construction
merger_tree_operators
merger_tree_outputters
merger_tree_read_importers
merger_tree_state_store
merger_tree_timesteps
merger_tree_walkers
merger_trees_build_mass_resolution
merger_trees_build_masses
merger_trees_build_masses_-distributions
merger_trees_builders
merger_trees_evolve
merger_trees_evolve_node
merger_trees_merge_node
meta_tree_compute_times
model_parameters
models_likelihoods
mpi_utilities
nbody_importers
nbody_operators
node_component_disk_standard
node_component_spheroid_standard
node_property_extractors
output_analyses
output_analysis_distribution_-normalizers
output_analysis_distribution_-operators
output_analysis_molecular_ratios
output_analysis_property_operators
output_analysis_weight_operators
output_times
posterior_sample_differential_-proposal_size
posterior_sample_differential_-random_jump
posterior_sampling_convergence
posterior_sampling_prop_size_temp_exp
posterior_sampling_simulation
posterior_sampling_state
posterior_sampling_state_initialize
posterior_sampling_stopping_criteria
power_spectra
power_spectra_nonlinear
power_spectra_primordial
power_spectra_primordial_transferred
power_spectrum_window_functions
radiation_fields
ram_pressure_stripping_mass_loss_-rate_disks
ram_pressure_stripping_mass_loss_-rate_spheroids
satellite_dynamical_friction
satellite_merging_mass_movements
satellite_merging_progenitor_-properties
satellite_merging_remnant_sizes
satellite_merging_timescales
satellite_oprhan_distributions
satellite_tidal_heating
satellite_tidal_stripping
satellites_tidal_fields
star_formation_feedback_disks
star_formation_feedback_expulsion_-disks
star_formation_feedback_expulsion_-spheroids
star_formation_feedback_spheroids

star_formation_histories
star_formation_rate_surface_density_disks
star_formation_timescales_disks
star_formation_timescales_spheroids
statistics_nbody_halo_mass_errors
stellar_astrophysics
stellar_astrophysics_tracks
stellar_astrophysics_winds
stellar_feedback
stellar_luminosities_structure
stellar_population_properties
stellar_population_selectors
stellar_population_spectra
stellar_population_spectra_postprocess
stellar_populations
stellar_populations_initial_mass_functions
stellar_spectra_dust_attenuations
string_handling
supernovae_population_iii
supernovae_type_ia
task_evolve_forests_work_shares
tasks
tidal_stripping_mass_loss_rate_disks
tidal_stripping_mass_loss_rate_spheroids
transfer_functions
unevolved_subhalo_mass_functions
universe_operators
virial_density_contrast
virial_orbits

subroutine: `state_initialize`

Description: Initialize the state module by getting the name of the file to which states should be stored and whether or not we are to retrieve a state.
Code lines: 36
Contained by: module galacticus_state
Modules used: input_parameters

file: `geometry.coordinate_systems.F90`

Description: Contains a module which implements calculations related to coordinate systems and transformations.
Code lines: 105

module: `coordinate_systems`

Description: Implements calculations related to coordinate systems and transformations.
Code lines: 83
Contained by: file geometry.coordinate_systems.F90
Used by: function galactic_structure_density
function galactic_structure_surface_density
program test_coordinate_systems

function: `coordinates_cartesian_to_cylindrical`

Description: Convert (x, y, z) in Cartesian coordinates into (r, ϕ, z) in cylindrical coordinates, with $\phi = 0$ corresponding to the x-axis.
Code lines: 13
Contained by: module coordinate_systems

function: `coordinates_cartesian_to_spherical`

Description: Convert (x, y, z) in Cartesian coordinates into (r, θ, ϕ) in spherical coordinates, with $\theta = 0$ corresponding to the z-axis and $\phi = 0$ corresponding to the x-axis.
Code lines: 20

Contained by: module coordinate_systems

function: coordinates_cylindrical_to_spherical
Description: Convert (R, ϕ, z) in cylindrical coordinates into (r, θ, ϕ) in spherical coordinates, with $\phi = 0$ corresponding to the x-axis.
Code lines: 20
Contained by: module coordinate_systems

function: coordinates_spherical_to_cylindrical
Description: Convert (r, θ, ϕ) in spherical coordinates into (R, ϕ, z) in cylindrical coordinates, with $\phi = 0$ corresponding to the x-axis.
Code lines: 14
Contained by: module coordinate_systems

file: geometry.lightcones.F90
Description: Contains a module which implements lightcone geometries.
Code lines: 69

module: geometry_lightcones
Description: Implements geometries of lightcones.
Code lines: 47
Contained by: file geometry.lightcones.F90
Modules used: galacticus_nodes, iso_c_binding
Used by: file galactic.filters.lightcone.F90, subroutine galacticus_function_classes_destroy, subroutine galacticus_state_retrieve, subroutine galacticus_state_store, file merger_trees.operators.prune_lightcone.F90, file nodes.property_extractor.lightcone.F90

file: geometry.lightcones.square.F90
Description: An implementation of the lightcone geometry class which assumes a square field of view.
Code lines: 718
Modules used: cosmology_functions

interface: geometrylightconesquare
Description: Constructors for the square dark matter halo spin distribution class.
Code lines: 4
Contained by: file geometry.lightcones.square.F90

function: squareconstructorinternal
Description: Internal constructor for the square lightcone geometry distribution class.
Code lines: 102
Contained by: file geometry.lightcones.square.F90
Modules used: galacticus_error, iso_varying_string, memory_management, numerical_constants_math, sort, string_handling, trigonometric_functions, vectors

function: `inversecosineintegral`
Description: Integral of $\sin(x) * \cos^{-1}[a/tan(x)]$ evaluated using Wolfram Alpha.
Code lines: 10
Contained by: function `squareconstructorinternal`

function: `squareconstructorparameters`
Description: Constructor for the `square` lightcone geometry distribution class which takes a parameter list as input.
Code lines: 125
Contained by: file `geometry.lightcones.square.F90`
Modules used: `cosmology_parameters`, `galacticus_error`, `input_parameters`, `numerical_constants_astronomical`

subroutine: `squaredestructor`
Description: Destructor for the `square` lightcone geometry distribution class.
Code lines: 7
Contained by: file `geometry.lightcones.square.F90`

function: `squareisinlightcone`
Description: Determine if the given `node` lies within the lightcone.
Code lines: 145
Contained by: file `geometry.lightcones.square.F90`
Modules used: `arrays_search`, `galacticus_error`, `galacticus_nodes`, `iso_c_binding`, `iso_varying_string`, `memory_management`, `numerical_comparison`, `string_handling`, `vectors`

function: `squareposition`
Description: Return the position of the node in lightcone coordinates.
Code lines: 36
Contained by: file `geometry.lightcones.square.F90`
Modules used: `arrays_search`, `galacticus_error`, `galacticus_nodes`, `iso_c_binding`, `iso_varying_string`, `numerical_comparison`, `string_handling`

function: `squarepositionatoutput`
Description: Return the position of the node in lightcone coordinates.
Code lines: 12
Contained by: file `geometry.lightcones.square.F90`
Modules used: `iso_c_binding`

subroutine: `squarereplicants`
Description: Compute quantities related to the number of replicants in which a node appears.
Code lines: 107
Contained by: file `geometry.lightcones.square.F90`

Modules used: galacticus_error iso_c_binding vectors

function: squarereplicationcount
Description: Determine the number of times node appears in the lightcone.
Code lines: 18
Contained by: file geometry.lightcones.square.F90
Modules used: arrays_search galacticus_nodes iso_c_binding

function: squaresolidangle
Description: Return the solid angle (in steradians) of a square lightcone.
Code lines: 7
Contained by: file geometry.lightcones.square.F90

function: squarevelocity
Description: Return the velocity of the node in lightcone coordinates.
Code lines: 19
Contained by: file geometry.lightcones.square.F90
Modules used: galacticus_nodes

file: geometry.mangle.F90
Description: Contains a module which implements functions utilizing MANGLE survey geometry definitions.
Code lines: 412

module: geometry_mangle
Description: Implements functions utilizing MANGLE survey geometry definitions.
Code lines: 390
Contained by: file geometry.mangle.F90
Modules used: iso_c_binding
Used by: file geometry.surveys.mangle.F90

type: cap
Description: A class to hold MANGLE caps.
Code lines: 15
Contained by: module geometry_mangle

function: cappointincluded
Description: Return true if a given Cartesian point lies within a MANGLE cap.
Code lines: 17
Contained by: module geometry_mangle
Modules used: vectors

function: geometrymangleangularpower
Description: Compute the angular power spectra of a MANGLE geometry.
Code lines: 113
Contained by: module geometry_mangle
Modules used: file_utilities galacticus_error

galacticus_paths io_hdf5
iso_varying_string numerical_constants_math
string_handling system_command

subroutine: geometrymanglebuild
Description: Download and build the MANGLE code.
Code lines: 22
Contained by: module geometry_mangle
Modules used: file_utilities galacticus_error
galacticus_paths iso_varying_string
system_command

function: geometrymanglesolidangle
Description: Compute the solid angle of a MANGLE geometry.
Code lines: 69
Contained by: module geometry_mangle
Modules used: file_utilities galacticus_error
galacticus_paths io_hdf5
iso_varying_string numerical_constants_math
string_handling system_command

type: polygon
Description: A class to hold MANGLE polygons.
Code lines: 16
Contained by: module geometry_mangle

function: polygonpointincluded
Description: Return true if a given Cartesian point lies within a MANGLE polygon, i.e. lies within *all* of the polygons caps.
Code lines: 14
Contained by: module geometry_mangle
Modules used: vectors

type: window
Description: A class to hold MANGLE windows.
Code lines: 23
Contained by: module geometry_mangle

function: windowpointincluded
Description: Return true if the given Cartesian point lies inside a MANGLE window, i.e. if it lies within any polygon of the window.
Code lines: 16
Contained by: module geometry_mangle

subroutine: windowread
Description: Read a MANGLE window definition from file.
Code lines: 57
Contained by: module geometry_mangle
Modules used: galacticus_display galacticus_error

iso_varying_string sort
string_handling

file: geometry.surveys.Baldry-2012-GAMA.F90
Description: Implements the geometry of the GAMA survey used by Baldry et al. [2012].
Code lines: 163
Modules used: cosmology_functions galacticus_paths

function: baldry2012gamaangularpowermaximumdegree
Description: Return the maximum degree for which angular power is computed for the Bernardi et al. [2013] survey.
Code lines: 8
Contained by: file geometry.surveys.Baldry-2012-GAMA.F90

function: baldry2012gamaconstructorinternal
Description: Internal constructor for the Baldry et al. [2012] conditional mass function class.
Code lines: 12
Contained by: file geometry.surveys.Baldry-2012-GAMA.F90
Modules used: cosmology_functions_options

function: baldry2012gamaconstructorparameters
Description: Constructor for the Baldry et al. [2012] conditional mass function class which takes a parameter set as input.
Code lines: 16
Contained by: file geometry.surveys.Baldry-2012-GAMA.F90
Modules used: cosmology_functions input_parameters

subroutine: baldry2012gamadestructor
Description: Destructor for the "baldry2012GAMA" survey geometry class.
Code lines: 7
Contained by: file geometry.surveys.Baldry-2012-GAMA.F90

function: baldry2012gamadistancemaximum
Description: Compute the maximum distance at which a galaxy is visible.
Code lines: 25
Contained by: file geometry.surveys.Baldry-2012-GAMA.F90
Modules used: galacticus_error

function: baldry2012gamafieldcount
Description: Return the number of fields in this sample.
Code lines: 8
Contained by: file geometry.surveys.Baldry-2012-GAMA.F90

function: baldry2012gamamangledirectory
Description: Return the path to the directory containing MANGLE files.
Code lines: 9
Contained by: file geometry.surveys.Baldry-2012-GAMA.F90

subroutine: baldry2012gamamanglefiles

Description: Return a list of MANGLE files.
Code lines: 9
Contained by: file `geometry.surveys.Baldry-2012-GAMA.F90`

interface: `surveygeometrybaldry2012gama`
Description: Constructors for the Baldry et al. [2012] survey geometry class.
Code lines: 4
Contained by: file `geometry.surveys.Baldry-2012-GAMA.F90`

file: `geometry.surveys.Bernardi-2013-SDSS.F90`
Description: Implements the geometry of the SDSS survey used by Bernardi et al. [2013].
Code lines: 170
Modules used: `galacticus_paths`

function: `bernardi2013sdssangularpowermaximumdegree`
Description: Return the maximum degree for which angular power is computed for the Bernardi et al. [2013] survey.
Code lines: 8
Contained by: file `geometry.surveys.Bernardi-2013-SDSS.F90`

function: `bernardi2013sdssconstructorinternal`
Description: Default constructor for the Bernardi et al. [2013] conditional mass function class.
Code lines: 8
Contained by: file `geometry.surveys.Bernardi-2013-SDSS.F90`
Modules used: `input_parameters`

function: `bernardi2013sdssconstructorparameters`
Description: Constructor for the Bernardi et al. [2013] conditional mass function class which takes a parameter set as input.
Code lines: 11
Contained by: file `geometry.surveys.Bernardi-2013-SDSS.F90`
Modules used: `input_parameters`

function: `bernardi2013sdssdistancemaximum`
Description: Compute the maximum distance at which a galaxy is visible.
Code lines: 15
Contained by: file `geometry.surveys.Bernardi-2013-SDSS.F90`
Modules used: `galacticus_error`

function: `bernardi2013sdssfieldcount`
Description: Return the number of fields in this sample.
Code lines: 8
Contained by: file `geometry.surveys.Bernardi-2013-SDSS.F90`

function: `bernardi2013sdssmangledirectory`
Description: Return the path to the directory containing MANGLE files.
Code lines: 9
Contained by: file `geometry.surveys.Bernardi-2013-SDSS.F90`

subroutine: `bernardi2013sdssmanglefiles`
Description: Return a list of MANGLE files.
Code lines: 23
Contained by: file `geometry.surveys.Bernardi-2013-SDSS.F90`
Modules used: `file_utilities` `galacticus_error` `system_command`

function: `bernardi2013sdsspointincluded`
Description: Return true if a point is included in the survey geometry.
Code lines: 26
Contained by: file `geometry.surveys.Bernardi-2013-SDSS.F90`
Modules used: `numerical_constants_units` `vectors`

interface: `surveygeometrybernardi2013sdss`
Description: Constructors for the Bernardi et al. [2013] survey geometry class.
Code lines: 4
Contained by: file `geometry.surveys.Bernardi-2013-SDSS.F90`

file: `geometry.surveys.Caputi-2011-UKIDSS-UDS.F90`
Description: Implements the survey geometry used by Caputi et al. [2011].
Code lines: 206
Modules used: `cosmology_functions`

function: `caputi2011ukidssudsconstructorinternal`
Description: Internal constructor for the Caputi et al. [2011] conditional mass function class.
Code lines: 28
Contained by: file `geometry.surveys.Caputi-2011-UKIDSS-UDS.F90`
Modules used: `cosmology_functions_options` `galacticus_error`

function: `caputi2011ukidssudsconstructorparameters`
Description: Default constructor for the Caputi et al. [2011] conditional mass function class.
Code lines: 22
Contained by: file `geometry.surveys.Caputi-2011-UKIDSS-UDS.F90`
Modules used: `input_parameters`

subroutine: `caputi2011ukidssudsdestructor`
Description: Destructor for the "caputi2011UKIDSSUDS" survey geometry class.
Code lines: 7
Contained by: file `geometry.surveys.Caputi-2011-UKIDSS-UDS.F90`

function: `caputi2011ukidssudsdistancemaximum`
Description: Compute the maximum distance at which a galaxy is visible.
Code lines: 22
Contained by: file `geometry.surveys.Caputi-2011-UKIDSS-UDS.F90`
Modules used: `cosmology_functions_options` `galacticus_error`

function: `caputi2011ukidssudsdistanceminimum`

Description: Compute the minimum distance at which a galaxy is included.
Code lines: 10
Contained by: file geometry.surveys.Caputi-2011-UKIDSS-UDS.F90

subroutine: caputi2011ukidssudsrandomsinitialize
Description: Load random points for the survey.
Code lines: 29
Contained by: file geometry.surveys.Caputi-2011-UKIDSS-UDS.F90
Modules used: file_utilities, galacticus_display, galacticus_error, galacticus_paths, io_hdf5, iso_varying_string, memory_management, numerical_constants_math, string_handling, system_command

function: caputi2011ukidssudssolidangle
Description: Return the solid angle of the Caputi et al. [2011] sample. Computed from survey mask (see constraints/dataAnalysis/stellarMassFunctions_UKIDSS_UDS_z3_-5/surveyGeometryRandoms.pl).
Code lines: 14
Contained by: file geometry.surveys.Caputi-2011-UKIDSS-UDS.F90
Modules used: galacticus_error

function: caputi2011ukidssudsvolumemaximum
Description: Compute the maximum volume within which a galaxy is visible.
Code lines: 13
Contained by: file geometry.surveys.Caputi-2011-UKIDSS-UDS.F90
Modules used: galacticus_error

interface: surveygeometrycaputi2011ukidssuds
Description: Constructors for the Caputi et al. [2011] survey geometry class.
Code lines: 4
Contained by: file geometry.surveys.Caputi-2011-UKIDSS-UDS.F90

file: geometry.surveys.Davidzon-2013-VIPERS.F90
Description: Implements the geometry of the VIPERS survey used by Davidzon et al. [2013].
Code lines: 218
Modules used: cosmology_functions, galacticus_paths

function: davidzon2013vipersangularpowermaximumdegree
Description: Return the maximum degree for which angular power is computed for the Davidzon et al. [2013] survey.
Code lines: 8
Contained by: file geometry.surveys.Davidzon-2013-VIPERS.F90

function: davidzon2013viperssconstructorinternal
Description: Generic constructor for the Davidzon et al. [2013] mass function class.
Code lines: 31
Contained by: file geometry.surveys.Davidzon-2013-VIPERS.F90
Modules used: cosmology_functions, cosmology_functions_options

galacticus_error input_parameters

function: davidzon2013vipersconstructorparameters
Description: Constructor for the Davidzon et al. [2013] conditional mass function class which takes a parameter set as input.
Code lines: 22
Contained by: file geometry.surveys.Davidzon-2013-VIPERS.F90
Modules used: input_parameters

subroutine: davidzon2013vipersdestructor
Description: Destructor for the "baldry2012GAMA" survey geometry class.
Code lines: 7
Contained by: file geometry.surveys.Davidzon-2013-VIPERS.F90

function: davidzon2013vipersdistancemaximum
Description: Compute the maximum distance at which a galaxy is visible.
Code lines: 27
Contained by: file geometry.surveys.Davidzon-2013-VIPERS.F90
Modules used: galacticus_error

function: davidzon2013vipersdistanceminimum
Description: Compute the minimum distance at which a galaxy is included.
Code lines: 10
Contained by: file geometry.surveys.Davidzon-2013-VIPERS.F90

function: davidzon2013vipersfieldcount
Description: Return the number of fields in this sample.
Code lines: 8
Contained by: file geometry.surveys.Davidzon-2013-VIPERS.F90

function: davidzon2013vipersmangledirectory
Description: Return the path to the directory containing MANGLE files.
Code lines: 9
Contained by: file geometry.surveys.Davidzon-2013-VIPERS.F90

subroutine: davidzon2013vipersmanglefiles
Description: Return a list of MANGLE files.
Code lines: 9
Contained by: file geometry.surveys.Davidzon-2013-VIPERS.F90

function: davidzon2013vipersvolumemaximum
Description: Compute the maximum volume within which a galaxy is visible.
Code lines: 11
Contained by: file geometry.surveys.Davidzon-2013-VIPERS.F90
Modules used: galacticus_error

interface: surveygeometrydavidzon2013vipers
Description: Constructors for the Davidzon et al. [2013] survey geometry class.
Code lines: 4

Contained by: file geometry.surveys.Davidzon-2013-VIPERS.F90

file: geometry.surveys.F90
Description: Contains a module which implements geometries of galaxy surveys.
Code lines: 116

module: geometry_surveys
Description: Implements geometries of galaxy surveys.
Code lines: 94
Contained by: file geometry.surveys.F90
Modules used: iso_c_binding
Used by:

file galactic.filters.survey_-geometry.F90	function surveygeometryconstructorparameters
subroutine galacticus_function_-classes_destroy	function localgroupmassfunctionconstructorinternal
file galacticus.output.analyses.correlation_-function.F90	file galacticus.output.analyses.luminosity_-function.F90
file galacticus.output.analyses.luminosity_-function.Halpha.F90	file galacticus.output.analyses.mass_-function_HI.F90
file galacticus.output.analyses.mass_-function_stellar.F90	function output_analysis_output_-weight_survey_volume
file galacticus.output.analyses.weight_-operator.cosmology.volume.F90	subroutine galacticus_state_retrieve
subroutine galacticus_state_store	subroutine halo_model_projected_-correlation
file models.likelihoods.mass_-function.F90	file models.likelihoods.projected_-correlation_function.F90
subroutine points_survey_geometry	file tasks.catalog_projected_-correlation_function.F90
file tasks.conditional_mass_-function.F90	file tasks.halo_model.projected_-correlation_function.F90
subroutine halomodelgenerateperform	file tasks.mass_function_covariance.F90

file: geometry.surveys.Gunawardhana-2013-SDSS.F90
Description: Implements the geometry of the SDSS survey used by Gunawardhana et al. [2013].
Code lines: 117
Modules used: cosmology_functions, galacticus_paths

function: gunawardhana2013sdssconstructorinternal
Description: Default constructor for the Gunawardhana et al. [2013] survey geometry class.
Code lines: 11
Contained by: file geometry.surveys.Gunawardhana-2013-SDSS.F90
Modules used: galacticus_error

function: gunawardhana2013sdssconstructorparameters

Description: Constructor for the Gunawardhana et al. [2013] conditional mass function class which takes a parameter set as input.
Code lines: 13
Contained by: file geometry.surveys.Gunawardhana-2013-SDSS.F90
Modules used: input_parameters

subroutine: gunawardhana2013sdssdestructor
Description: Destructor for the "gunawardhana2013SDSS" survey geometry class.
Code lines: 7
Contained by: file geometry.surveys.Gunawardhana-2013-SDSS.F90

function: gunawardhana2013sdssdistancemaximum
Description: Compute the maximum distance at which a galaxy is visible.
Code lines: 23
Contained by: file geometry.surveys.Gunawardhana-2013-SDSS.F90
Modules used: cosmology_functions_options galacticus_error numerical_constants_astronomical numerical_constants_units

function: gunawardhana2013sdssdistanceminimum
Description: Compute the minimum distance at which a galaxy is visible.
Code lines: 11
Contained by: file geometry.surveys.Gunawardhana-2013-SDSS.F90

interface: surveygeometrygunawardhana2013sdss
Description: Constructors for the Gunawardhana et al. [2013] survey geometry class.
Code lines: 4
Contained by: file geometry.surveys.Gunawardhana-2013-SDSS.F90

file: geometry.surveys.Hearin-2014-SDSS.F90
Description: Implements the survey geometry of the SDSS sample used by Hearin et al. [2013].
Code lines: 116
Modules used: cosmology_functions

function: hearin2014sdssconstructorinternal
Description: Internal constructor for the Hearin et al. [2013] conditional mass function class.
Code lines: 17
Contained by: file geometry.surveys.Hearin-2014-SDSS.F90
Modules used: cosmology_functions_options galacticus_error

function: hearin2014sdssconstructorparameters
Description: Default constructor for the Hearin et al. [2013] conditional mass function class.
Code lines: 15
Contained by: file geometry.surveys.Hearin-2014-SDSS.F90
Modules used: input_parameters

subroutine: hearin2014sdssdestructor
Description: Destructor for the "hearin2014SDSS" survey geometry class.
Code lines: 7
Contained by: file geometry.surveys.Hearin-2014-SDSS.F90

function: `hearin2014sdssdistancemaximum`
Description: Compute the maximum distance at which a galaxy is visible.
Code lines: 11
Contained by: file `geometry.surveys.Hearin-2014-SDSS.F90`

function: `hearin2014sdssdistanceminimum`
Description: Compute the minimum distance at which a galaxy is visible.
Code lines: 10
Contained by: file `geometry.surveys.Hearin-2014-SDSS.F90`

interface: `surveygeometryhearin2014sdss`
Description: Constructors for the Hearin et al. [2013] survey geometry class.
Code lines: 4
Contained by: file `geometry.surveys.Hearin-2014-SDSS.F90`

file: `geometry.surveys.Kelvin-2014-GAMAnear.F90`
Description: Implements the geometry of the GAMAnear survey used by Kelvin et al. [2014].
Code lines: 105
Modules used: `galacticus_paths`

function: `kelvin2014gamanearconstructorinternal`
Description: Internal constructor for the Kelvin et al. [2014] conditional mass function class.
Code lines: 14
Contained by: file `geometry.surveys.Kelvin-2014-GAMAnear.F90`
Modules used: `cosmology_functions_options`

function: `kelvin2014gamanearconstructorparameters`
Description: Default constructor for the Kelvin et al. [2014] conditional mass function class.
Code lines: 15
Contained by: file `geometry.surveys.Kelvin-2014-GAMAnear.F90`
Modules used: `input_parameters`

function: `kelvin2014gamaneardistancemaximum`
Description: Compute the maximum distance at which a galaxy is visible.
Code lines: 17
Contained by: file `geometry.surveys.Kelvin-2014-GAMAnear.F90`
Modules used: `galacticus_error`

function: `kelvin2014gamaneardistanceminimum`
Description: Compute the minimum distance at which a galaxy is included in the survey.
Code lines: 11
Contained by: file `geometry.surveys.Kelvin-2014-GAMAnear.F90`
Modules used: `galacticus_error`

interface: `surveygeometrykelvin2014gamanear`
Description: Constructors for the Kelvin et al. [2014] survey geometry class.
Code lines: 4
Contained by: file `geometry.surveys.Kelvin-2014-GAMAnear.F90`

file: geometry.surveys.Li-White-2009-SDSS.F90
Description: Implements the survey geometry of the SDSS sample used by Li and White [2009].
Code lines: 210
Modules used: cosmology_functions

function: liwhite2009sdssconstructorinternal
Description: Constructor for the Li and White [2009] survey geometry class which allows specification of minimum and maximum redshifts.
Code lines: 18
Contained by: file geometry.surveys.Li-White-2009-SDSS.F90
Modules used: cosmology_functions cosmology_functions_options

function: liwhite2009sdssconstructorparameters
Description: Constructor for the Li and White [2009] survey geometry class which takes a parameter set as input.
Code lines: 32
Contained by: file geometry.surveys.Li-White-2009-SDSS.F90
Modules used: input_parameters

subroutine: liwhite2009sdssdestructor
Description: Destructor for the "liWhite2009SDSS" survey geometry class.
Code lines: 7
Contained by: file geometry.surveys.Li-White-2009-SDSS.F90

function: liwhite2009sdssdistancemaximum
Description: Compute the maximum distance at which a galaxy is visible.
Code lines: 20
Contained by: file geometry.surveys.Li-White-2009-SDSS.F90
Modules used: cosmology_functions_options galacticus_error

function: liwhite2009sdssdistanceminimum
Description: Compute the minimum distance at which a galaxy is visible.
Code lines: 10
Contained by: file geometry.surveys.Li-White-2009-SDSS.F90

subroutine: liwhite2009sdssrandomsinitialize
Description: Compute the window function for the survey.
Code lines: 51
Contained by: file geometry.surveys.Li-White-2009-SDSS.F90
Modules used: file_utilities galacticus_display
galacticus_error galacticus_paths
iso_varying_string memory_management
numerical_constants_math string_handling
system_command

function: liwhite2009sdsssolidangle
Description: Return the solid angle of the Li and White [2009] sample.
Code lines: 13
Contained by: file geometry.surveys.Li-White-2009-SDSS.F90

Modules used: galacticus_error

interface: surveygeometryliwhite2009sdss
Description: Constructors for the Li and White [2009] survey geometry class.
Code lines: 4
Contained by: file geometry.surveys.Li-White-2009-SDSS.F90

file: geometry.surveys.Local_Group_DES.F90
Description: Implements the geometry of the DES survey for Local Group dwarfs.
Code lines: 139
Modules used: galacticus_paths

function: localgroupdesangularpowermaximumdegree
Description: Return the maximum degree for which angular power is computed for the localGroupDES survey.
Code lines: 8
Contained by: file geometry.surveys.Local_Group_DES.F90

function: localgroupdesconstructorinternal
Description: Internal constructor for the localGroupDES survey geometry class.
Code lines: 9
Contained by: file geometry.surveys.Local_Group_DES.F90

function: localgroupdesconstructorparameters
Description: Constructor for the localGroupDES conditional mass function class which takes a parameter set as input.
Code lines: 19
Contained by: file geometry.surveys.Local_Group_DES.F90
Modules used: input_parameters

function: localgroupdesdistancemaximum
Description: Compute the maximum distance at which a galaxy is visible.
Code lines: 17
Contained by: file geometry.surveys.Local_Group_DES.F90
Modules used: galacticus_error

function: localgroupdesfieldcount
Description: Return the number of fields in this sample.
Code lines: 8
Contained by: file geometry.surveys.Local_Group_DES.F90

function: localgroupdesmangledirectory
Description: Return the path to the directory containing MANGLE files.
Code lines: 9
Contained by: file geometry.surveys.Local_Group_DES.F90

subroutine: localgroupdesmanglefiles
Description: Return a list of MANGLE files.
Code lines: 9

Contained by: file geometry.surveys.Local_Group_DES.F90

interface: surveygeometrylocalgroupdes
Description: Constructors for the localGroupDES survey geometry class.
Code lines: 4
Contained by: file geometry.surveys.Local_Group_DES.F90

file: geometry.surveys.Local_Group_SDSS.F90
Description: Implements the geometry of the SDSS survey with a depth for Local Group dwarf detection.
Code lines: 86

function: localgroupsdssconstructorinternal
Description: Internal constructor for the Local Group SDSS survey geometry class
Code lines: 9
Contained by: file geometry.surveys.Local_Group_SDSS.F90

function: localgroupsdssconstructorparameters
Description: Constructor for the Local Group SDSS survey geometry class which takes a parameter set as input.
Code lines: 19
Contained by: file geometry.surveys.Local_Group_SDSS.F90
Modules used: input_parameters

function: localgroupsdsssdistancemaximum
Description: Compute the maximum distance at which a galaxy is visible.
Code lines: 15
Contained by: file geometry.surveys.Local_Group_SDSS.F90
Modules used: galacticus_error

interface: surveygeometrylocalgroupsdss
Description: Constructors for the Bernardi et al. [2013] survey geometry class.
Code lines: 4
Contained by: file geometry.surveys.Local_Group_SDSS.F90

file: geometry.surveys.Local_Group_classical.F90
Description: Implements a geometry corresponding to the detectability of classical Local Group galaxies.
Code lines: 150
Modules used: galacticus_paths

function: localgroupclassicalangularpowermaximumdegree
Description: Return the maximum degree for which angular power is computed for the Local Group Classical galaxies survey.
Code lines: 8
Contained by: file geometry.surveys.Local_Group_classical.F90

function: localgroupclassicalconstructorinternal
Description: Internal constructor for the Baldry et al. [2012] conditional mass function class.
Code lines: 9
Contained by: file geometry.surveys.Local_Group_classical.F90

function: `localgroupclassicalconstructorparameters`
Description: Constructor for the `localGroupClassical` survey geometry class which takes a parameter set as input.
Code lines: 27
Contained by: file `geometry.surveys.Local_Group_classical.F90`
Modules used: `input_parameters`

function: `localgroupclassicaldistancemaximum`
Description: Compute the maximum distance at which a galaxy is visible.
Code lines: 16
Contained by: file `geometry.surveys.Local_Group_classical.F90`

function: `localgroupclassicalfieldcount`
Description: Return the number of fields in this sample.
Code lines: 8
Contained by: file `geometry.surveys.Local_Group_classical.F90`

function: `localgroupclassicalmangledirectory`
Description: Return the path to the directory containing MANGLE files.
Code lines: 9
Contained by: file `geometry.surveys.Local_Group_classical.F90`

subroutine: `localgroupclassicalmanglefiles`
Description: Return a list of MANGLE files.
Code lines: 9
Contained by: file `geometry.surveys.Local_Group_classical.F90`

interface: `surveygeometrylocalgroupclassical`
Description: Constructors for the Baldry et al. [2012] survey geometry class.
Code lines: 4
Contained by: file `geometry.surveys.Local_Group_classical.F90`

file: `geometry.surveys.Martin-2010-ALFALFA.F90`
Description: Implements the survey geometry used by Martin et al. [2010].
Code lines: 200
Modules used: `cosmology_parameters`

function: `martin2010alfalfaconstructorinternal`
Description: Internal constructor for the Martin et al. [2010] conditional mass function class.
Code lines: 10
Contained by: file `geometry.surveys.Martin-2010-ALFALFA.F90`
Modules used: `input_parameters`

function: `martin2010alfalfaconstructorparameters`
Description: Constructor for the Martin et al. [2010] conditional mass function class which takes a parameter list as input.
Code lines: 15
Contained by: file `geometry.surveys.Martin-2010-ALFALFA.F90`
Modules used: `input_parameters`

subroutine: martin2010alfalfadestructor
Description: Destructor for the "martin2010ALFALFA" survey geometry class.
Code lines: 7
Contained by: file geometry.surveys.Martin-2010-ALFALFA.F90

function: martin2010alfalfadistancemaximum
Description: Compute the maximum distance at which a galaxy is visible.
Code lines: 38
Contained by: file geometry.surveys.Martin-2010-ALFALFA.F90
Modules used: cosmology_parameters, galacticus_error

subroutine: martin2010alfalfarandomsinitialize
Description: Initialize random points for the survey.
Code lines: 60
Contained by: file geometry.surveys.Martin-2010-ALFALFA.F90
Modules used: cosmology_functions, cosmology_parameters, file_utilities, galacticus_display, galacticus_error, galacticus_paths, iso_c_binding, iso_varying_string, memory_management, meshes, numerical_constants_astronomical, numerical_constants_math, pseudo_random, string_handling, system_command, vectors

function: martin2010alfalfasolidangle
Description: Return the solid angle of the Martin et al. [2010] sample.
Code lines: 13
Contained by: file geometry.surveys.Martin-2010-ALFALFA.F90
Modules used: galacticus_error

interface: surveygeometrymartin2010alfalfa
Description: Constructors for the Martin et al. [2010] survey geometry class.
Code lines: 4
Contained by: file geometry.surveys.Martin-2010-ALFALFA.F90

file: geometry.surveys.Montero-Dorta-2009-SDSS.F90
Description: Implements the geometry of the SDSS survey used by Montero-Dorta and Prada [2009].
Code lines: 169
Modules used: cosmology_functions, galacticus_paths

function: monterodorta2009sdssconstructorinternal
Description: Default constructor for the Montero-Dorta and Prada [2009] survey geometry class.
Code lines: 46
Contained by: file geometry.surveys.Montero-Dorta-2009-SDSS.F90
Modules used: galacticus_error

function: monterodorta2009sdssconstructorparameters
Description: Constructor for the Montero-Dorta and Prada [2009] conditional mass function class which takes a parameter set as input.

Code lines: 21
Contained by: file geometry.surveys.Montero-Dorta-2009-SDSS.F90
Modules used: input_parameters

subroutine: monterodorta2009sdssdestructor
Description: Destructor for the "monteroDorta2009SDSS" survey geometry class.
Code lines: 7
Contained by: file geometry.surveys.Montero-Dorta-2009-SDSS.F90

function: monterodorta2009sdssdistancemaximum
Description: Compute the maximum distance at which a galaxy is visible.
Code lines: 26
Contained by: file geometry.surveys.Montero-Dorta-2009-SDSS.F90
Modules used: cosmology_functions_options galacticus_error

function: monterodorta2009sdssdistanceminimum
Description: Compute the maximum distance at which a galaxy is visible.
Code lines: 15
Contained by: file geometry.surveys.Montero-Dorta-2009-SDSS.F90
Modules used: cosmology_functions_options galacticus_error

interface: surveygeometrymonterodorta2009sdss
Description: Constructors for the Montero-Dorta and Prada [2009] survey geometry class.
Code lines: 4
Contained by: file geometry.surveys.Montero-Dorta-2009-SDSS.F90

file: geometry.surveys.Moustakas-2013-PRIMUS.F90
Description: Implements the geometry of the PRIMUS survey used by Moustakas et al. [2013].
Code lines: 252
Modules used: cosmology_functions galacticus_paths

function: moustakas2013primusangularpowermaximumdegree
Description: Return the maximum degree for which angular power is computed for the Moustakas et al. [2013] survey.
Code lines: 8
Contained by: file geometry.surveys.Moustakas-2013-PRIMUS.F90

function: moustakas2013primusconstructorinternal
Description: Generic constructor for the Moustakas et al. [2013] mass function class.
Code lines: 42
Contained by: file geometry.surveys.Moustakas-2013-PRIMUS.F90
Modules used: cosmology_functions cosmology_functions_options galacticus_error input_parameters

function: moustakas2013primusconstructorparameters
Description: Default constructor for the Moustakas et al. [2013] conditional mass function class.
Code lines: 22
Contained by: file geometry.surveys.Moustakas-2013-PRIMUS.F90
Modules used: input_parameters

subroutine: `moustakas2013primusdestructor`
Description: Destructor for the "moustakas2013PRIMUS" survey geometry class.
Code lines: 7
Contained by: file `geometry.surveys.Moustakas-2013-PRIMUS.F90`

function: `moustakas2013primusdistancemaximum`
Description: Compute the maximum distance at which a galaxy is visible.
Code lines: 36
Contained by: file `geometry.surveys.Moustakas-2013-PRIMUS.F90`
Modules used: `cosmology_functions_options` `galacticus_error`

function: `moustakas2013primusdistanceminimum`
Description: Compute the minimum distance at which a galaxy is included.
Code lines: 10
Contained by: file `geometry.surveys.Moustakas-2013-PRIMUS.F90`

function: `moustakas2013primusfieldcount`
Description: Return the number of fields in this sample.
Code lines: 8
Contained by: file `geometry.surveys.Moustakas-2013-PRIMUS.F90`

function: `moustakas2013primusfieldpairindex`
Description: Compute the index of a pair of fields in the Moustakas et al. [2013] survey.
Code lines: 10
Contained by: file `geometry.surveys.Moustakas-2013-PRIMUS.F90`

function: `moustakas2013primusmangledirectory`
Description: Return the path to the directory containing MANGLE files.
Code lines: 9
Contained by: file `geometry.surveys.Moustakas-2013-PRIMUS.F90`

subroutine: `moustakas2013primusmanglefiles`
Description: Return a list of MANGLE files.
Code lines: 9
Contained by: file `geometry.surveys.Moustakas-2013-PRIMUS.F90`

function: `moustakas2013primusvolumemaximum`
Description: Compute the maximum volume within which a galaxy is visible.
Code lines: 14
Contained by: file `geometry.surveys.Moustakas-2013-PRIMUS.F90`
Modules used: `galacticus_error`

interface: `surveygeometrymoustakas2013primus`
Description: Constructors for the Moustakas et al. [2013] survey geometry class.
Code lines: 4
Contained by: file `geometry.surveys.Moustakas-2013-PRIMUS.F90`

file: `geometry.surveys.Muzzin-2013-ULTRAVISTA.F90`
Description: Implements the geometry of the ULTRAVISTA survey used by Muzzin et al. [2013].
Code lines: 231
Modules used: `cosmology_functions` `galacticus_paths`

function: `muzzin2013ultravistaangularpowermaximumdegree`
Description: Return the maximum degree for which angular power is computed for the Muzzin et al. [2013] survey.
Code lines: 8
Contained by: file `geometry.surveys.Muzzin-2013-ULTRAVISTA.F90`

function: `muzzin2013ultravistaconstructorinternal`
Description: Internal constructor for the Muzzin et al. [2013] mass function class.
Code lines: 41
Contained by: file `geometry.surveys.Muzzin-2013-ULTRAVISTA.F90`
Modules used: `cosmology_functions_options` `galacticus_error`

function: `muzzin2013ultravistaconstructorparameters`
Description: Default constructor for the Muzzin et al. [2013] conditional mass function class.
Code lines: 22
Contained by: file `geometry.surveys.Muzzin-2013-ULTRAVISTA.F90`
Modules used: `input_parameters`

subroutine: `muzzin2013ultravistadestructor`
Description: Destructor for the "muzzin2013ULTRAVISTA" survey geometry class.
Code lines: 7
Contained by: file `geometry.surveys.Muzzin-2013-ULTRAVISTA.F90`

function: `muzzin2013ultravistadistancemaximum`
Description: Compute the maximum distance at which a galaxy is visible.
Code lines: 29
Contained by: file `geometry.surveys.Muzzin-2013-ULTRAVISTA.F90`
Modules used: `cosmology_functions_options` `galacticus_error`

function: `muzzin2013ultravistadistanceminimum`
Description: Compute the minimum distance at which a galaxy is included.
Code lines: 10
Contained by: file `geometry.surveys.Muzzin-2013-ULTRAVISTA.F90`

function: `muzzin2013ultravistafieldcount`
Description: Return the number of fields in this sample.
Code lines: 8
Contained by: file `geometry.surveys.Muzzin-2013-ULTRAVISTA.F90`

function: `muzzin2013ultravistamangledirectory`
Description: Return the path to the directory containing MANGLE files.
Code lines: 9
Contained by: file `geometry.surveys.Muzzin-2013-ULTRAVISTA.F90`

subroutine: `muzzin2013ultravistamanglefiles`
Description: Return a list of MANGLE files.
Code lines: 10
Contained by: file `geometry.surveys.Muzzin-2013-ULTRAVISTA.F90`

function: `muzzin2013ultravistavolumemaximum`
Description: Compute the maximum volume within which a galaxy is visible.
Code lines: 11
Contained by: file `geometry.surveys.Muzzin-2013-ULTRAVISTA.F90`
Modules used: `galacticus_error`

interface: `surveygeometrymuzzin2013ultravista`
Description: Constructors for the Muzzin et al. [2013] survey geometry class.
Code lines: 4
Contained by: file `geometry.surveys.Muzzin-2013-ULTRAVISTA.F90`

file: `geometry.surveys.Tomczak-2014-ZFOURGE.F90`
Description: Implements the geometry of the ZFOURGE survey used by Tomczak et al. [2014].
Code lines: 233
Modules used: `cosmology_functions` `galacticus_paths`

interface: `surveygeometrytomczak2014zfourge`
Description: Constructors for the Tomczak et al. [2014] survey geometry class.
Code lines: 4
Contained by: file `geometry.surveys.Tomczak-2014-ZFOURGE.F90`

function: `tomczak2014zfourgeangularpowermaximumdegree`
Description: Return the maximum degree for which angular power is computed for the Tomczak et al. [2014] survey.
Code lines: 8
Contained by: file `geometry.surveys.Tomczak-2014-ZFOURGE.F90`

function: `tomczak2014zfourgeconstructorinternal`
Description: Generic constructor for the Tomczak et al. [2014] mass function class.
Code lines: 43
Contained by: file `geometry.surveys.Tomczak-2014-ZFOURGE.F90`
Modules used: `cosmology_functions_options` `galacticus_error`

function: `tomczak2014zfourgeconstructorparameters`
Description: Default constructor for the Tomczak et al. [2014] conditional mass function class.
Code lines: 22
Contained by: file `geometry.surveys.Tomczak-2014-ZFOURGE.F90`
Modules used: `input_parameters`

subroutine: `tomczak2014zfourgedestructor`
Description: Destructor for the "tomczak2014ZFOURGE" survey geometry class.
Code lines: 7
Contained by: file `geometry.surveys.Tomczak-2014-ZFOURGE.F90`

function: `tomczak2014zfourgedistancemaximum`
Description: Compute the maximum distance at which a galaxy is visible.
Code lines: 30
Contained by: file `geometry.surveys.Tomczak-2014-ZFOURGE.F90`
Modules used: `cosmology_functions_options` `galacticus_error`

function: `tomczak2014zfourgedistanceminimum`
Description: Compute the minimum distance at which a galaxy is included.
Code lines: 10
Contained by: file `geometry.surveys.Tomczak-2014-ZFOURGE.F90`

function: `tomczak2014zfourgefieldcount`
Description: Return the number of fields in this sample.
Code lines: 8
Contained by: file `geometry.surveys.Tomczak-2014-ZFOURGE.F90`

function: `tomczak2014zfourgemangledirectory`
Description: Return the path to the directory containing MANGLE files.
Code lines: 9
Contained by: file `geometry.surveys.Tomczak-2014-ZFOURGE.F90`

subroutine: `tomczak2014zfourgemanglefiles`
Description: Return a list of MANGLE files.
Code lines: 9
Contained by: file `geometry.surveys.Tomczak-2014-ZFOURGE.F90`

function: `tomczak2014zfourgevolumemaximum`
Description: Compute the maximum volume within which a galaxy is visible.
Code lines: 11
Contained by: file `geometry.surveys.Tomczak-2014-ZFOURGE.F90`
Modules used: `galacticus_error`

file: `geometry.surveys.combined.F90`
Description: Implements a survey geometry which combines multiple other surveys.
Code lines: 220

function: `combinedangularpower`
Description: Return the survey angular power C_ℓ^{ij} from MANGLE polygons.
Code lines: 11
Contained by: file `geometry.surveys.combined.F90`
Modules used: `galacticus_error`

function: `combinedangularpoweravailable`
Description: Return false to indicate that survey angular power is not available.
Code lines: 8
Contained by: file `geometry.surveys.combined.F90`

function: `combinedconstructorinternal`
Description: Internal constructor for the combined output analysis weight operator class.
Code lines: 14
Contained by: file geometry.surveys.combined.F90

function: `combinedconstructorparameters`
Description: Constructor for the "combined" survey geometry class which takes a parameter set as input.
Code lines: 22
Contained by: file geometry.surveys.combined.F90
Modules used: input_parameters

subroutine: `combineddeepcopy`
Description: Perform a deep copy for the `combined` surveyGeometry class.
Code lines: 31
Contained by: file geometry.surveys.combined.F90
Modules used: galacticus_error

subroutine: `combineddestructor`
Description: Destructor for the combined weight operator class.
Code lines: 16
Contained by: file geometry.surveys.combined.F90

function: `combinedpointincluded`
Description: Return true if a point is included in the combined survey geometry.
Code lines: 17
Contained by: file geometry.surveys.combined.F90
Modules used: galacticus_error

function: `combinedsolidangle`
Description: Return the survey solid angle.
Code lines: 11
Contained by: file geometry.surveys.combined.F90
Modules used: galacticus_error

function: `combinedwindowfunctionavailable`
Description: Return false to indicate that survey window function is not available.
Code lines: 8
Contained by: file geometry.surveys.combined.F90

subroutine: `combinedwindowfunctions`
Description: Provides window functions for combined survey geometries.
Code lines: 14
Contained by: file geometry.surveys.combined.F90
Modules used: galacticus_error iso_c_binding

interface: `surveygeometrycombined`
Code lines: 3
Contained by: file geometry.surveys.combined.F90

type: `surveygeometrylist`
Code lines: 3
Contained by: file `geometry.surveys.combined.F90`

file: `geometry.surveys.full_sky.F90`
Description: Implements survey geometries over the full sky.
Code lines: 268
Modules used: `cosmology_functions`

function: `fullskyangularpower`
Description: Return the angular power for the full sky.
Code lines: 14
Contained by: file `geometry.surveys.full_sky.F90`
Modules used: `numerical_constants_math`

function: `fullskyangularpoweravailable`
Description: Return true to indicate that survey angular power is available.
Code lines: 8
Contained by: file `geometry.surveys.full_sky.F90`

function: `fullskyconstructorinternal`
Description: Constructor for the full sky survey class which allows specification of minimum and maximum redshifts.
Code lines: 16
Contained by: file `geometry.surveys.full_sky.F90`
Modules used: `cosmology_functions_options`

function: `fullskyconstructorparameters`
Description: Default constructor for the full sky survey geometry.
Code lines: 32
Contained by: file `geometry.surveys.full_sky.F90`
Modules used: `input_parameters`

subroutine: `fullskydestructor`
Description: Destructor for the "fullSky" survey geometry class.
Code lines: 7
Contained by: file `geometry.surveys.full_sky.F90`

function: `fullskydistancemaximum`
Description: Compute the maximum distance at which a galaxy is visible.
Code lines: 10
Contained by: file `geometry.surveys.full_sky.F90`

function: `fullskydistanceminimum`
Description: Compute the minimum distance at which a galaxy is visible.
Code lines: 10
Contained by: file `geometry.surveys.full_sky.F90`

function: `fullskypointincluded`
Description: Return true if a point is included in the survey geometry.
Code lines: 13
Contained by: file `geometry.surveys.full_sky.F90`
Modules used: `vectors`

function: `fullskysolidangle`
Description: Return the solid angle of the Li and White [2009] sample.
Code lines: 10
Contained by: file `geometry.surveys.full_sky.F90`
Modules used: `numerical_constants_math`

function: `fullskywindowfunctionavailable`
Description: Return true to indicate that survey window function is available.
Code lines: 8
Contained by: file `geometry.surveys.full_sky.F90`

subroutine: `fullskywindowfunctions`
Description: Compute the window function for the survey.
Code lines: 69
Contained by: file `geometry.surveys.full_sky.F90`
Modules used: `fftw3`, `galacticus_display`, `galacticus_error`, `iso_c_binding`, `iso_varying_string`, `memory_management`, `meshes`, `numerical_constants_math`, `pseudo_random`, `string_handling`, `vectors`

interface: `surveygeometryfullsky`
Description: Constructors for the full sky survey geometry class.
Code lines: 4
Contained by: file `geometry.surveys.full_sky.F90`

file: `geometry.surveys.mangle.F90`
Description: Implements an abstract survey geometry using MANGLE polygons.
Code lines: 255
Modules used: `galacticus_paths`, `geometry_mangle`

function: `mangleangularpower`
Description: Return the survey angular power C_ℓ^{ij} from MANGLE polygons.
Code lines: 37
Contained by: file `geometry.surveys.mangle.F90`
Modules used: `file_utilities`, `galacticus_error`, `io_hdf5`, `memory_management`, `string_handling`, `system_command`

function: `mangleangularpoweravailable`

Description: Return true to indicate that survey angular power is available.
Code lines: 8
Contained by: file geometry.surveys.mangle.F90

function: manglefieldpairindex
Description: Compute the index of a pair of fields in MANGLE-based survey geometries.
Code lines: 11
Contained by: file geometry.surveys.mangle.F90

subroutine: mangleinitialize
Description: Internal constructor for the MANGLE conditional mass function class.
Code lines: 9
Contained by: file geometry.surveys.mangle.F90

function: manglepointincluded
Description: Return true if a point is included in the survey geometry.
Code lines: 28
Contained by: file geometry.surveys.mangle.F90
Modules used: galacticus_error vectors

function: manglesolidangle
Description: Return the survey solid angle computed from MANGLE polygons.
Code lines: 35
Contained by: file geometry.surveys.mangle.F90
Modules used: galacticus_error string_handling

function: manglewindowfunctionavailable
Description: Return false to indicate that survey window function is not available.
Code lines: 8
Contained by: file geometry.surveys.mangle.F90

subroutine: manglewindowfunctions
Description: Provides window functions for MANGLE-based survey geometries.
Code lines: 14
Contained by: file geometry.surveys.mangle.F90
Modules used: galacticus_error iso_c_binding

type: surveygeometrymangle
Code lines: 36
Contained by: file geometry.surveys.mangle.F90

file: geometry.surveys.random_points.F90
Description: Implements survey geometries defined by random points.
Code lines: 176

function: randompointincluded
Description: Return true if a point is included in the survey geometry.
Code lines: 12
Contained by: file geometry.surveys.random_points.F90

Modules used: galacticus_error

function: randompointsangularpower
Description: Angular power is not available, so simply aborts.
Code lines: 11
Contained by: file geometry.surveys.random_points.F90
Modules used: galacticus_error

function: randompointsangularpoweravailable
Description: Return false to indicate that survey angular power is not available.
Code lines: 8
Contained by: file geometry.surveys.random_points.F90

function: randompointswindowfunctionavailable
Description: Return true to indicate that survey window function is available.
Code lines: 8
Contained by: file geometry.surveys.random_points.F90

subroutine: randompointswindowfunctions
Description: Compute the window function for the survey.
Code lines: 73
Contained by: file geometry.surveys.random_points.F90
Modules used: fftw3, galacticus_display, galacticus_error, iso_c_binding, memory_management, meshes, numerical_constants_math, pseudo_random, string_handling, vectors

type: surveygeometryrandompoints
Code lines: 19
Contained by: file geometry.surveys.random_points.F90

file: halo_model.conditional_mass_function.Behroozi2010.F90
Description: Implements a class for the conditional mass functions using the Behroozi et al. [2010] fitting function.
Code lines: 370
Modules used: tables

subroutine: behroozi2010compute
Description: Computes the cumulative conditional mass function, $\langle N(M_\star|M_{\rm halo})\rangle \equiv \phi(M_\star|M_{\rm halo})$ using the fitting formula of Behroozi et al. [2010].
Code lines: 63
Contained by: file halo_model.conditional_mass_function.Behroozi2010.F90
Modules used: table_labels

function: behroozi2010constructorinternal
Description: Internal constructor for the Behroozi et al. [2010] conditional mass function class.
Code lines: 14
Contained by: file halo_model.conditional_mass_function.Behroozi2010.F90

function: `behroozi2010constructorparameters`
Description: Constructor for the `behroozi2010` conditional mass function class which builds the object from a parameter set.
Code lines: 122
Contained by: file `halo_model.conditional_mass_function.Behroozi2010.F90`
Modules used: `input_parameters`

subroutine: `behroozi2010destructor`
Description: Destructor for the Behroozi et al. [2010] conditional mass function class.
Code lines: 8
Contained by: file `halo_model.conditional_mass_function.Behroozi2010.F90`

function: `behroozi2010fshmrinverse`
Description: The median mass vs. halo mass relation functional form from Behroozi et al. [2010].
Code lines: 17
Contained by: file `halo_model.conditional_mass_function.Behroozi2010.F90`

function: `behroozi2010massfunction`
Description: Compute the cumulative conditional mass function, $\langle N(M_\star|M_{\rm halo})\rangle \equiv \phi(M_\star|M_{\rm halo})$ using the fitting formula of Behroozi et al. [2010].
Code lines: 33
Contained by: file `halo_model.conditional_mass_function.Behroozi2010.F90`
Modules used: `galacticus_error`

function: `behroozi2010massfunctionvariance`
Description: Computes the variance in the cumulative conditional mass function, $\langle N(M_\star|M_{\rm halo})\rangle \equiv \phi(M_\star|M_{\rm halo})$ using the fitting formula of Behroozi et al. [2010]. Assumes that the number of satellite galaxies is Poisson distributed, while the number of central galaxies follows a Bernoulli distribution, and that the numbers of satellites and centrals are uncorrelated.
Code lines: 18
Contained by: file `halo_model.conditional_mass_function.Behroozi2010.F90`

interface: `conditionalmassfunctionbehroozi2010`
Description: Constructors for the Behroozi et al. [2010] merging timescale class.
Code lines: 4
Contained by: file `halo_model.conditional_mass_function.Behroozi2010.F90`

file: `halo_model.conditional_mass_function.F90`
Description: Contains a module which implements a class for empirical models of conditional mass functions.
Code lines: 55

module: `conditional_mass_functions`
Description: Implements empirical models of conditional mass functions.
Code lines: 33
Contained by: file `halo_model.conditional_mass_function.F90`
Used by: subroutine `galacticus_function_-classes_destroy` subroutine `galacticus_state_retrieve`

subroutine galacticus_state_store
subroutine halo_model_projected_-correlation
file merger_-trees.construct.build.masses.distribution.stellar_-mass_function.F90
function massfunctionevaluate
function projectedcorrelationfunctionevaluate
file tasks.conditional_mass_-function.F90
file tasks.halo_model.projected_-correlation_function.F90
file tasks.halo_model_generate.F90
subroutine halomodelgenerateperform
file tasks.mass_function_covariance.F90

file: halo_model.power_spectrum_modifier.F90
Description: Contains a module which implements a class of modifiers of the power spectrum for the halo model.
Code lines: 51

module: halo_model_power_spectrum_modifiers
Description: Implements a class of modifiers of the power spectrum for the halo model.
Code lines: 29
Contained by: file halo_model.power_spectrum_modifier.F90
Used by: subroutine galacticus_function_-classes_destroy
file galacticus.output.analyses.correlation_-function.F90
subroutine correlationfunctionaccumulatehalo
subroutine galacticus_state_retrieve
subroutine galacticus_state_store

file: halo_model.power_spectrum_modifier.identity.F90
Description: Implements a identity modifier for power spectra in the halo model of clustering.
Code lines: 64

interface: halomodelpowerspectrummodifieridentity
Description: Constructors for the identity halo model power spectrum modifier class.
Code lines: 3
Contained by: file halo_model.power_spectrum_modifier.identity.F90

function: identityconstructorparameters
Description: Default constructor for the identity hot halo outflow reincorporation class which takes a parameter set as input.
Code lines: 11
Contained by: file halo_model.power_spectrum_modifier.identity.F90
Modules used: input_parameters

subroutine: identitymodify
Description: Applies a identity modification to a halo model power spectrum.
Code lines: 14
Contained by: file halo_model.power_spectrum_modifier.identity.F90

file: halo_model.power_spectrum_modifier.triaxiality.F90

Description: Implements a triaxiality modifier for power spectra in the halo model of clustering based on the results of Smith and Watts [2005].
Code lines: 147
Modules used: cosmology_parameters tables

interface: halomodelpowerspectrummodifiertriaxiality
Description: Constructors for the triaxiality halo model power spectrum modifier class.
Code lines: 4
Contained by: file halo_model.power_spectrum_modifier.triaxiality.F90

function: triaxialityconstructorinternal
Description: Default constructor for the triaxiality hot halo outflow reincorporation class.
Code lines: 16
Contained by: file halo_model.power_spectrum_modifier.triaxiality.F90
Modules used: galacticus_error table_labels

function: triaxialityconstructorparameters
Description: Default constructor for the triaxiality hot halo outflow reincorporation class which takes a parameter set as input.
Code lines: 14
Contained by: file halo_model.power_spectrum_modifier.triaxiality.F90
Modules used: input_parameters

subroutine: triaxialitydestructor
Description: Destructor for the triaxiality halo model power spectrum modifier class.
Code lines: 7
Contained by: file halo_model.power_spectrum_modifier.triaxiality.F90

subroutine: triaxialitymodify
Description: Applies a triaxiality modification to a halo model power spectrum based on the results of Smith and Watts [2005].
Code lines: 52
Contained by: file halo_model.power_spectrum_modifier.triaxiality.F90
Modules used: galacticus_error vectors

file: halo_model.projected_correlation_function.F90
Description: Contains a module which implements calculations of projected correlation functions using the halo model.
Code lines: 332

module: halo_model_projected_correlations
Description: Implements calculations of projected correlation functions using the halo model.
Code lines: 310
Contained by: file halo_model.projected_correlation_function.F90
Used by:

function	subroutine
projectedcorrelationfunctionevaluate	halomodelprojectedcorrelationfunctionperform

subroutine: halo_model_projected_correlation
Description: Compute the projected correlation function of galaxies above a specified mass using the halo model.

Code lines: 295
Contained by: module halo_model_projected_correlations
Modules used: conditional_mass_functions, cosmology_functions, dark_matter_halo_biases, dark_matter_halo_scales, dark_matter_profile_scales, dark_matter_profiles_dmo, fftlogs, fgsl, galacticus_error, galacticus_nodes, geometry_surveys, halo_mass_functions, linear_growth, memory_management, node_component_dark_matter_profile_scale, numerical_constants_math, numerical_integration, numerical_ranges, power_spectra, table_labels, tables

function: binningintegrandweight
Description: The weight function applied to the projected correlation function when integrating into bins.
Code lines: 7
Contained by: subroutine halo_model_projected_correlation

function: normalizationintegrand
Description: Integrand for the normalization term in the power spectrum.
Code lines: 7
Contained by: subroutine halo_model_projected_correlation

function: normalizationtimeintegrand
Description: Time integrand for the normalization term in the power spectrum.
Code lines: 13
Contained by: subroutine halo_model_projected_correlation

function: powerspectrumonehalointegrand
Description: Integrand for the one-halo term in the power spectrum.
Code lines: 26
Contained by: subroutine halo_model_projected_correlation
Modules used: galacticus_calculations_resets

function: powerspectrumonehalotimeintegrand
Description: Time integrand for the one-halo term in the power spectrum.
Code lines: 22
Contained by: subroutine halo_model_projected_correlation
Modules used: galacticus_display

function: powerspectrumtwohalointegrand
Description: Integrand for the two-halo term in the power spectrum.
Code lines: 20
Contained by: subroutine halo_model_projected_correlation
Modules used: galacticus_calculations_resets

function: `powerspectrumtwohalotimeintegrand`
Description: Time integrand for the two-halo term in the power spectrum.
Code lines: 22
Contained by: subroutine `halo_model_projected_correlation`
Modules used: `galacticus_display`

function: `projectionintegrandweight`
Description: The weight function applied to the correlation function when integrating to get the projected correlation function.
Code lines: 11
Contained by: subroutine `halo_model_projected_correlation`

function: `volumetimeintegrand`
Description: Volume integrand for the normalization term in the power spectrum.
Code lines: 8
Contained by: subroutine `halo_model_projected_correlation`

file: `hdf5FCInterop.F90`
Description: Contains a program which determine C interoperable types corresponding to HDF5 types.
Code lines: 70

program: `hdf5fcinterop`
Description: Determine C interoperable types corresponding to HDF5 types. This allows us to avoid compiler warnings about possible C non-interoperability if we were to use the types provided directly by HDF5.
Code lines: 44
Contained by: file `hdf5FCInterop.F90`
Modules used: `hdf5` `iso_c_binding`

file: `hiiRegions.emission_lines.F90`
Description: Contains a module that provides functions for emission line calculations.
Code lines: 87

module: `hii_region_emission_lines`
Code lines: 63
Contained by: file `hiiRegions.emission_lines.F90`
Modules used: `iso_varying_string`
Used by: subroutine `filter_response_load` subroutine `stellar_luminosities_initialize` subroutine `stellar_luminosities_special_cases`

subroutine: `emissionlinedatabaseinitialize`
Description: Initialize a database of emission line properties.
Code lines: 28
Contained by: module `hii_region_emission_lines`
Modules used: `galacticus_paths` `io_hdf5`

function: `emissionlinewavelength`
Description: Return the wavelength of a named emission line.
Code lines: 19
Contained by: module hii_region_emission_lines
Modules used: galacticus_error

file: `hot_halo.cold_mode.density_profile.core_radius.F90`
Description: Contains a module which provides a class that implements core radii for cored cold mode hot halo mass distributions.
Code lines: 40

module: `hot_halo_cold_mode_density_core_radii`
Description: Provides a module which provides a class that implements core radii for cored cold mode hot halo mass distributions.
Code lines: 18
Contained by: file hot_halo.cold_mode.density_profile.core_radius.F90
Modules used: galacticus_nodes
Used by: subroutine galacticus_function_classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; subroutine node_component_hot_halo_cold_mode_thread_initialize; module node_component_hot_halo_cold_mode_structure_tasks

file: `hot_halo.cold_mode.density_profile.core_radius.virial_radius.F90`
Description: Implements a cold mode hot halo mass distribution core radius class which sets the core radius to a fraction of the virial radius.
Code lines: 100
Modules used: dark_matter_halo_scales

interface: `hothalocoldmodecoreradiivirialfraction`
Description: Constructors for the `virialRadiusFraction` hot halo mass distribution core radius class.
Code lines: 4
Contained by: file hot_halo.cold_mode.density_profile.core_radius.virial_radius.F90

function: `virialradiusfractionconstructorinternal`
Description: Default constructor for the `virialRadiusFraction` cold mode hot halo mass distribution core radius class.
Code lines: 10
Contained by: file hot_halo.cold_mode.density_profile.core_radius.virial_radius.F90
Modules used: input_parameters

function: `virialradiusfractionconstructorparameters`
Description: A constructor for the `virialRadiusFraction` cold mode hot halo mass distribution core radius class which builds the object from a parameter set.
Code lines: 23
Contained by: file hot_halo.cold_mode.density_profile.core_radius.virial_radius.F90
Modules used: input_parameters

subroutine: `virialradiusfractiondestructor`

Description: Destructor for the `virialRadiusFraction` cold mode hot halo mass distribution core radius class.
Code lines: 7
Contained by: file `hot_halo.cold_mode.density_profile.core_radius.virial_radius.F90`

function: `virialradiusfractionradius`
Description: Return the core radius of the hot halo mass distribution.
Code lines: 8
Contained by: file `hot_halo.cold_mode.density_profile.core_radius.virial_radius.F90`

file: `hot_halo.mass_distribution.Enzo_hydrostatic.F90`
Description: An implementation of the hot halo mass distribution class which uses the "hydrostatic" profile used by the Enzo simulation code.
Code lines: 279
Modules used: `hot_halo_mass_distributions_core_radii` `hot_halo_temperature_profiles`

function: `enzohydrostaticconstructorinternal`
Description: Generic constructor for the `enzoHydrostatic` hot halo mass distribution class.
Code lines: 11
Contained by: file `hot_halo.mass_distribution.Enzo_hydrostatic.F90`
Modules used: `array_utilities` `galacticus_error`

function: `enzohydrostaticconstructorparameters`
Description: Default constructor for the `enzoHydrostatic` hot halo mass distribution class.
Code lines: 16
Contained by: file `hot_halo.mass_distribution.Enzo_hydrostatic.F90`
Modules used: `input_parameters`

function: `enzohydrostaticdensity`
Description: Return the density in a `enzoHydrostatic` hot halo mass distribution.
Code lines: 13
Contained by: file `hot_halo.mass_distribution.Enzo_hydrostatic.F90`

function: `enzohydrostaticdensitylogslope`
Description: Return the logarithmic slope of the density profile in a `enzoHydrostatic` hot halo mass distribution.
Code lines: 16
Contained by: file `hot_halo.mass_distribution.Enzo_hydrostatic.F90`

function: `enzohydrostaticdensitynormalization`
Description: Return the density normalization in a `enzoHydrostatic` hot halo mass distribution.
Code lines: 29
Contained by: file `hot_halo.mass_distribution.Enzo_hydrostatic.F90`
Modules used: `fgsl` `galacticus_nodes` `numerical_integration`

subroutine: `enzohydrostaticdestructor`
Description: Destructor for the `enzoHydrostatic` hot halo mass distribution class.
Code lines: 8

Contained by: file hot_halo.mass_distribution.Enzo_hydrostatic.F90

function: enzohydrostaticenclosedmass
Description: Return the enclosed mass in a enzoHydrostatic hot halo mass distribution.
Code lines: 30
Contained by: file hot_halo.mass_distribution.Enzo_hydrostatic.F90
Modules used: fgsl galacticus_nodes numerical_integration

function: enzohydrostaticenclosedmassintegrand
Description: Integrand used in finding the normalization of the enzoHydrostatic hot halo mass distribution.
Code lines: 15
Contained by: file hot_halo.mass_distribution.Enzo_hydrostatic.F90
Modules used: numerical_constants_math

function: enzohydrostaticradialmoment
Description: Return a radial moment of an enzoHydrostatic hot halo mass distribution.
Code lines: 45
Contained by: file hot_halo.mass_distribution.Enzo_hydrostatic.F90
Modules used: fgsl galacticus_nodes iso_c_binding numerical_integration

function: enzohydrostaticradialmomentintegrand
Description: Integrand used in finding the normalization of the enzoHydrostatic hot halo mass distribution.
Code lines: 15
Contained by: function enzohydrostaticradialmoment
Modules used: hot_halo_temperature_profiles

function: enzohydrostaticrotationnormalization
Description: Return the relation between specific angular momentum and rotation velocity (assuming a rotation velocity that is constant in radius) for node. Specifically, the normalization, A, returned is such that $V_{\rm rot} = AJ/M$.
Code lines: 13
Contained by: file hot_halo.mass_distribution.Enzo_hydrostatic.F90
Modules used: galacticus_nodes

interface: hothalomassdistributionenzohydrostatic
Description: Constructors for the enzoHydrostatic hot halo mass distribution class.
Code lines: 4
Contained by: file hot_halo.mass_distribution.Enzo_hydrostatic.F90

file: hot_halo.mass_distribution.F90
Description: Contains a module which provides a hot halo mass distribution class.
Code lines: 176

module: hot_halo_mass_distributions
Description: Provides an object which provides a hot halo mass distribution class.

Code lines:	154	
Contained by:	file hot_halo.mass_distribution.F90	
Modules used:	galacticus_nodes	
Used by:	file cooling.cooling_radius.beta_profile.F90	function betaprofileradius
	function betaprofileradiusgrowthrate	file cooling.cooling_radius.isothermal_profile.F90
	file cooling.cooling_radius.simple.F90	file cooling.cooling_rate.Cole2000.F90
	file cooling.cooling_rate.White-Frenk.F90	file cooling.specific_angular_momentum.constant_rotation.F90
	subroutine adiabaticgnedin2004computefactors	function galactic_structure_density
	function galactic_structure_enclosed_mass	function galactic_structure_rotation_curve
	function galactic_structure_rotation_curve_gradient	subroutine galacticus_function_classes_destroy
	subroutine galacticus_state_retrieve	subroutine galacticus_state_store
	file hot_halo.ram_pressure_force.Font2008.F90	file hot_halo.ram_pressure_stripping.Font2008.F90
	file hot_halo.ram_pressure_stripping.timescale.ram_pressure_acceleration.F90	file nodes.property_extractor.ICM_SZ.F90
	file nodes.property_extractor.ICM_Xray_luminosity.F90	module node_component_hot_halo_standard

function: `hothalomassdistributiondensity`

Description:	Computes the density at a given position for a dark matter profile.
Code lines:	18
Contained by:	module hot_halo_mass_distributions
Modules used:	galactic_structure_options

function: `hothalomassdistributionenclosedmass`

Description:	Computes the mass within a given radius for a dark matter profile.	
Code lines:	26	
Contained by:	module hot_halo_mass_distributions	
Modules used:	galactic_structure_options	galacticus_nodes

function: `hothalomassdistributionrotationcurve`

Description:	Computes the rotation curve at a given radius for the hot halo density profile.	
Code lines:	17	
Contained by:	module hot_halo_mass_distributions	
Modules used:	galactic_structure_options	numerical_constants_physical

function: `hothalomassdistributionrotationcurvegradient`

Description:	Computes the rotation curve gradient at a given radius for the hot halo density profile.	
Code lines:	24	
Contained by:	module hot_halo_mass_distributions	
Modules used:	galactic_structure_options	numerical_constants_math
	numerical_constants_physical	

file: `hot_halo.mass_distribution.PatejLoeb2015.F90`
Description: An implementation of the hot halo mass distribution class which uses the model of Patej and Loeb [2015].
Code lines: 248
Modules used: `dark_matter_halo_scales` `dark_matter_profiles_dmo`

interface: `hothalomassdistributionpatejloeb2015`
Description: Constructors for the `patejLoeb2015` hot halo mass distribution class.
Code lines: 4
Contained by: file `hot_halo.mass_distribution.PatejLoeb2015.F90`

function: `patejloeb2015constructorinternal`
Description: Generic constructor for the `patejLoeb2015` hot halo mass distribution class.
Code lines: 24
Contained by: file `hot_halo.mass_distribution.PatejLoeb2015.F90`
Modules used: `array_utilities` `galacticus_error` `galacticus_nodes`

function: `patejloeb2015constructorparameters`
Description: Default constructor for the `patejLoeb2015` hot halo mass distribution class.
Code lines: 33
Contained by: file `hot_halo.mass_distribution.PatejLoeb2015.F90`
Modules used: `input_parameters`

function: `patejloeb2015density`
Description: Return the density in a `patejLoeb2015` hot halo mass distribution.
Code lines: 23
Contained by: file `hot_halo.mass_distribution.PatejLoeb2015.F90`
Modules used: `galacticus_nodes`

function: `patejloeb2015densitylogslope`
Description: Return the density in a `patejLoeb2015` hot halo mass distribution.
Code lines: 17
Contained by: file `hot_halo.mass_distribution.PatejLoeb2015.F90`
Modules used: `galacticus_nodes`

subroutine: `patejloeb2015destructor`
Description: Destructor for the `patejLoeb2015` hot halo mass distribution class.
Code lines: 8
Contained by: file `hot_halo.mass_distribution.PatejLoeb2015.F90`

function: `patejloeb2015enclosedmass`
Description: Return the enclosed mass in a `patejLoeb2015` hot halo mass distribution.
Code lines: 26
Contained by: file `hot_halo.mass_distribution.PatejLoeb2015.F90`
Modules used: `galacticus_nodes`

function: `patejloeb2015radialmoment`

Description: Compute a radial moment in a `patejLoeb2015` hot halo mass distribution. For this profile we have:

$$\rho_{\rm g}(r) = f\Gamma(r/s)^{3\Gamma-3}\rho_{\rm DM}(s[r/s]^\Gamma). \tag{18.11}$$

Defining $R = s[r/s]^\Gamma$, such that $r/s = (R/s)^{1/\Gamma}$, and $\mathrm{d}r = \Gamma^{-1}(R/s)^{1/\Gamma-1}\mathrm{d}R$, then

$$\int r^m \rho_{\rm g}(r)\mathrm{d}r = f s^{(m-2)(\Gamma-1)/\Gamma} \int R^{(2\Gamma-2+m)/\Gamma}\rho_{\rm DM}(R)\mathrm{d}R, \tag{18.12}$$

or

$$\mathcal{R}_{\rm g}(r;m) = f s^{(m-2)(\Gamma-1)/\Gamma}\mathcal{R}_{\rm DM}(r;(2\Gamma-2+m)/\Gamma), \tag{18.13}$$

where $\mathcal{R}(r;m)$ is the $m^{\rm th}$ radial moment of the density profile.

Code lines: 39
Contained by: file `hot_halo.mass_distribution.PatejLoeb2015.F90`
Modules used: `galacticus_nodes`

function: `patejloeb2015rotationnormalization`

Description: Returns the relation between specific angular momentum and rotation velocity (assuming a rotation velocity that is constant in radius) for `node`. Specifically, the normalization, A, returned is such that $V_{\rm rot} = AJ/M$.

Code lines: 13
Contained by: file `hot_halo.mass_distribution.PatejLoeb2015.F90`
Modules used: `galacticus_nodes`

file: `hot_halo.mass_distribution.Ricotti2000.F90`

Description: An implementation of the hot halo mass distribution class which uses the model of Ricotti and Shull [2000].

Code lines: 134
Modules used: `dark_matter_halo_scales` `dark_matter_profiles_dmo`

interface: `hothalomassdistributionricotti2000`

Description: Constructors for the `ricotti2000` hot halo mass distribution class.

Code lines: 4
Contained by: file `hot_halo.mass_distribution.Ricotti2000.F90`

function: `ricotti2000constructorinternal`

Description: Internal constructor for the `ricotti2000` hot halo mass distribution class.

Code lines: 27
Contained by: file `hot_halo.mass_distribution.Ricotti2000.F90`
Modules used: `array_utilities` `galacticus_error` `galacticus_nodes`

function: `ricotti2000constructorparameters`

Description: Default constructor for the `ricotti2000` hot halo mass distribution class.

Code lines: 16
Contained by: file `hot_halo.mass_distribution.Ricotti2000.F90`
Modules used: `input_parameters`

subroutine: `ricotti2000destructor`
Description: Destructor for the `ricotti2000` hot halo mass distribution class.
Code lines: 8
Contained by: file `hot_halo.mass_distribution.Ricotti2000.F90`

subroutine: `ricotti2000initialize`
Description: Initialize the `ricotti2000` hot halo density profile for the given `node`. Parameterizations of β and core radius are taken from section 2.1 of Ricotti and Shull [2000].
Code lines: 31
Contained by: file `hot_halo.mass_distribution.Ricotti2000.F90`
Modules used: `galacticus_nodes`

file: `hot_halo.mass_distribution.beta_profile.F90`
Description: An implementation of the hot halo mass distribution class for β-profile distributions.
Code lines: 219
Modules used: `hot_halo_mass_distributions_core_radii` `mass_distributions`

function: `betaprofileconstructorinternal`
Description: Internal constructor for the `betaProfile` hot halo mass distribution class.
Code lines: 9
Contained by: file `hot_halo.mass_distribution.beta_profile.F90`

function: `betaprofileconstructorparameters`
Description: Constructor for the null `betaProfile` hot halo mass distributionclass which builds the object from a parameter set.
Code lines: 37
Contained by: file `hot_halo.mass_distribution.beta_profile.F90`
Modules used: `array_utilities` `galacticus_error` `galacticus_nodes` `input_parameters`

function: `betaprofiledensity`
Description: Return the density in a single-betaProfile hot halo mass distribution.
Code lines: 13
Contained by: file `hot_halo.mass_distribution.beta_profile.F90`
Modules used: `coordinates`

function: `betaprofiledensitylogslope`
Description: Return the logarithmic slope of the density of the hot halo at the given `radius`.
Code lines: 13
Contained by: file `hot_halo.mass_distribution.beta_profile.F90`
Modules used: `coordinates`

subroutine: `betaprofiledestructor`
Description: Destructor for the `betaProfile` hot halo mass distribution class.
Code lines: 7
Contained by: file `hot_halo.mass_distribution.beta_profile.F90`

function: `betaprofileenclosedmass`
Description: Return the mass enclosed in the hot halo at the given `radius`.
Code lines: 17
Contained by: file hot_halo.mass_distribution.beta_profile.F90
Modules used: galacticus_nodes

subroutine: `betaprofileinitialize`
Description: Initialize the β-profile hot halo density profile for the given `node`.
Code lines: 20
Contained by: file hot_halo.mass_distribution.beta_profile.F90
Modules used: galacticus_nodes

function: `betaprofileradialmoment`
Description: Return the radial moment of the density profile of the hot halo to the given `radius`.
Code lines: 13
Contained by: file hot_halo.mass_distribution.beta_profile.F90
Modules used: galacticus_nodes

function: `betaprofilerotationnormalization`
Description: Returns the relation between specific angular momentum and rotation velocity (assuming a rotation velocity that is constant in radius) for `node`. Specifically, the normalization, A, returned is such that $V_{\rm rot} = AJ/M$.
Code lines: 13
Contained by: file hot_halo.mass_distribution.beta_profile.F90
Modules used: galacticus_nodes

interface: `hothalomassdistributionbetaprofile`
Description: Constructors for the β-profile hot halo mass distribution class.
Code lines: 4
Contained by: file hot_halo.mass_distribution.beta_profile.F90

file: `hot_halo.mass_distribution.cored.core_radius.F90`
Description: Contains a module which provides an object that implements core radii for cored hot halo mass distributions.
Code lines: 40

module: `hot_halo_mass_distributions_core_radii`
Description: Provides an object that implements core radii for hot halo mass distributions.
Code lines: 18
Contained by: file hot_halo.mass_distribution.cored.core_radius.F90
Modules used: galacticus_nodes
Used by: subroutine galacticus_function_classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; file hot_halo.mass_distribution.Enzo_hydrostatic.F90; file hot_halo.mass_distribution.beta_profile.F90

file: `hot_halo.mass_distribution.cored.core_radius.growing.F90`
Description: An implementation of the hot halo mass distribution core radius class in which the core grows as the hot halo content is depleted.
Code lines: 186
Modules used: `cosmology_parameters` `dark_matter_halo_scales` `tables`

function: `growingconstructorinternal`
Description: Default constructor for the `growing` hot halo mass distribution core radius class.
Code lines: 14
Contained by: file `hot_halo.mass_distribution.cored.core_radius.growing.F90`
Modules used: `galacticus_error` `galacticus_nodes`

function: `growingconstructorparameters`
Description: Constructor for the `growing` hot halo mass distribution core radius class which builds the object from a parameter set.
Code lines: 34
Contained by: file `hot_halo.mass_distribution.cored.core_radius.growing.F90`
Modules used: `input_parameters`

function: `growingcorevirialdensityfunction`
Description: Returns the function $(1+r_c^2)[1-r_c \tan^{-1}(1/r_c]$ which is proportional to the density at the virial radius of a cored isothermal profile with core radius r_c (in units of the virial radius) per unit mass.
Code lines: 8
Contained by: file `hot_halo.mass_distribution.cored.core_radius.growing.F90`

subroutine: `growingdestructor`
Description: Destructor for the `growing` hot halo mass distribution class.
Code lines: 9
Contained by: file `hot_halo.mass_distribution.cored.core_radius.growing.F90`

function: `growingradius`
Description: Return the core radius of the hot halo mass distribution.
Code lines: 57
Contained by: file `hot_halo.mass_distribution.cored.core_radius.growing.F90`
Modules used: `galacticus_nodes`

interface: `hothalomassdistributioncoreradiusgrowing`
Description: Constructors for the `growing` hot halo mass distribution core radius class.
Code lines: 4
Contained by: file `hot_halo.mass_distribution.cored.core_radius.growing.F90`

file: `hot_halo.mass_distribution.cored.core_radius.virial_radius_fraction.F90`
Description: Implements a hot halo mass distribution core radius class which sets the core radius to a fraction of the virial radius.
Code lines: 99
Modules used: `dark_matter_halo_scales`

interface: `hothalomassdistributioncoreradiusvirialfraction`
Description: Constructors for the `virialFraction` hot halo mass distribution core radius class.
Code lines: 4
Contained by: file `hot_halo.mass_distribution.cored.core_radius.virial_radius_fraction.F90`

function: `virialfractionconstructorinternal`
Description: Default constructor for the `virialFraction` hot halo mass distribution core radius class.
Code lines: 10
Contained by: file `hot_halo.mass_distribution.cored.core_radius.virial_radius_fraction.F90`
Modules used: `input_parameters`

function: `virialfractionconstructorparameters`
Description: A constructor for the `virialFraction` hot halo mass distribution core radius class which builds the object from a parameter set.
Code lines: 23
Contained by: file `hot_halo.mass_distribution.cored.core_radius.virial_radius_fraction.F90`
Modules used: `input_parameters`

subroutine: `virialfractiondestructor`
Description: Destructor for the `virialFraction` hot halo mass distribution core radius class.
Code lines: 7
Contained by: file `hot_halo.mass_distribution.cored.core_radius.virial_radius_fraction.F90`

function: `virialfractionradius`
Description: Return the core radius of the hot halo mass distribution.
Code lines: 8
Contained by: file `hot_halo.mass_distribution.cored.core_radius.virial_radius_fraction.F90`

file: `hot_halo.mass_distribution.null.F90`
Description: A null implementation of the hot halo mass distribution class.
Code lines: 115

interface: `hothalomassdistributionnull`
Description: Constructors for the null hot halo mass distribution class.
Code lines: 3
Contained by: file `hot_halo.mass_distribution.null.F90`

function: `nullconstructorparameters`
Description: Constructor for the null hot halo mass distribution class which builds the object from a parameter set.
Code lines: 10
Contained by: file `hot_halo.mass_distribution.null.F90`
Modules used: `input_parameters`

function: `nulldensity`
Description: Return the density in a null hot halo mass distribution.
Code lines: 10
Contained by: file `hot_halo.mass_distribution.null.F90`

function: `nulldensitylogslope`
Description: Return the logarithmic slope of the density of the hot halo at the given `radius`.
Code lines: 10
Contained by: file `hot_halo.mass_distribution.null.F90`

function: `nullenclosedmass`
Description: Return the mass enclosed in the hot halo at the given `radius`.
Code lines: 10
Contained by: file `hot_halo.mass_distribution.null.F90`

function: `nullradialmoment`
Description: Return the density of the hot halo at the given `radius`.
Code lines: 10
Contained by: file `hot_halo.mass_distribution.null.F90`

function: `nullrotationnormalization`
Description: Returns the relation between specific angular momentum and rotation velocity (assuming a rotation velocity that is constant in radius) for `node`. Specifically, the normalization, A, returned is such that $V_{\rm rot} = AJ/M$.
Code lines: 11
Contained by: file `hot_halo.mass_distribution.null.F90`

file: `hot_halo.outflow_reincorporation.F90`
Description: Contains a module which provides a class that implements reincorporation of outflowed mass into the hot halo.
Code lines: 41

module: `hot_halo_outflows_reincorporations`
Description: Provides a class that implements reincorporation of outflowed mass into the hot halo.
Code lines: 19
Contained by: file `hot_halo.outflow_reincorporation.F90`
Modules used: `galacticus_nodes`
Used by: subroutine `galacticus_function_classes_destroy`; subroutine `galacticus_state_retrieve`; subroutine `galacticus_state_store`; module `node_component_hot_halo_standard`; subroutine `node_component_hot_halo_vs_delayed_rate_compute`

file: `hot_halo.outflow_reincorporation.Henriques2013.F90`
Description: An implementation of the hot halo outflow reincorporation class in which implements the model of Henriques et al. [2013].
Code lines: 135
Modules used: `cosmology_functions`; `dark_matter_halo_scales`

function: `henriques2013constructorinternal`
Description: Default constructor for the `henriques2013` hot halo outflow reincorporation class.
Code lines: 10

Contained by: file hot_halo.outflow_reincorporation.Henriques2013.F90

function: henriques2013constructorparameters
Description: Constructor for the henriques2013 hot halo outflow reincorporation class which takes a parameter set as input.
Code lines: 44
Contained by: file hot_halo.outflow_reincorporation.Henriques2013.F90
Modules used: input_parameters

subroutine: henriques2013destructor
Description: Destructor for the henriques2013 hot halo outflow reincorporation class.
Code lines: 8
Contained by: file hot_halo.outflow_reincorporation.Henriques2013.F90

function: henriques2013rate
Description: Return the rate of mass reincorporation for outflowed gas in the hot halo.
Code lines: 14
Contained by: file hot_halo.outflow_reincorporation.Henriques2013.F90
Modules used: galacticus_nodes

interface: hothalooutflowreincorporationhenriques2013
Description: Constructors for the henriques2013 hot halo outflow reincorporation class.
Code lines: 4
Contained by: file hot_halo.outflow_reincorporation.Henriques2013.F90

file: hot_halo.outflow_reincorporation.halo_dynamical_time.F90
Description: An implementation of the hot halo outflow reincorporation class in which reincorporation occurs on a multiple of the halo dynamical timescale.
Code lines: 102
Modules used: dark_matter_halo_scales

function: halodynamicaltimeconstructorinternal
Description: Default constructor for the haloDynamicalTime hot halo outflow reincorporation class.
Code lines: 9
Contained by: file hot_halo.outflow_reincorporation.halo_dynamical_time.F90

function: halodynamicaltimeconstructorparameters
Description: Default constructor for the haloDynamicalTime hot halo outflow reincorporation class which takes a parameter set as input.
Code lines: 23
Contained by: file hot_halo.outflow_reincorporation.halo_dynamical_time.F90
Modules used: input_parameters

subroutine: halodynamicaltimedestructor
Description: Destructor for the haloDynamicalTime hot halo outflow reincorporation class.
Code lines: 7
Contained by: file hot_halo.outflow_reincorporation.halo_dynamical_time.F90

function: halodynamicaltimerate

Description: Return the rate of mass reincorporation for outflowed gas in the hot halo.
Code lines: 11
Contained by: file hot_halo.outflow_reincorporation.halo_dynamical_time.F90
Modules used: galacticus_nodes

interface: hothalooutflowreincorporationhalodynamicaltime
Description: Constructors for the haloDynamicalTime hot halo outflow reincorporation class.
Code lines: 4
Contained by: file hot_halo.outflow_reincorporation.halo_dynamical_time.F90

file: hot_halo.outflow_reincorporation.velocity_maximum_scaling.F90
Description: An implementation of the hot halo outflow reincorporation class which uses simple scalings based on the halo maximum circular velocity.
Code lines: 217
Modules used: cosmology_functions, dark_matter_profiles_dmo, kind_numbers, math_exponentiation

interface: hothalooutflowreincorporationvelocitymaximumscaling
Description: Constructors for the velocityMaximumScaling hot halo outflow reincorporation class.
Code lines: 4
Contained by: file hot_halo.outflow_reincorporation.velocity_maximum_scaling.F90

subroutine: velocitymaximumscalingautohook
Description: Attach to the calculation reset event.
Code lines: 8
Contained by: file hot_halo.outflow_reincorporation.velocity_maximum_scaling.F90
Modules used: events_hooks

subroutine: velocitymaximumscalingcalculationreset
Description: Reset the halo scales calculation.
Code lines: 11
Contained by: file hot_halo.outflow_reincorporation.velocity_maximum_scaling.F90

function: velocitymaximumscalingconstructorinternal
Description: Default constructor for the velocityMaximumScaling hot halo outflow reincorporation class.
Code lines: 26
Contained by: file hot_halo.outflow_reincorporation.velocity_maximum_scaling.F90
Modules used: galacticus_error, galacticus_nodes

function: velocitymaximumscalingconstructorparameters
Description: Default constructor for the velocityMaximumScaling hot halo outflow reincorporation class which takes a parameter set as input.
Code lines: 50
Contained by: file hot_halo.outflow_reincorporation.velocity_maximum_scaling.F90
Modules used: input_parameters

subroutine: velocitymaximumscalingdestructor
Description: Destructor for the Galacticus format merger tree importer class.
Code lines: 10

Contained by: file hot_halo.outflow_reincorporation.velocity_maximum_scaling.F90
Modules used: events_hooks

function: velocitymaximumscalingrate
Description: Return the rate of mass reincorporation for outflowed gas in the hot halo.
Code lines: 34
Contained by: file hot_halo.outflow_reincorporation.velocity_maximum_scaling.F90
Modules used: galacticus_nodes

file: hot_halo.outflow_reincorporation.zero.F90
Description: An implementation of the hot halo outflow reincorporation class which gives zero reincorporation rate.
Code lines: 61

interface: hothalooutflowreincorporationzero
Description: Constructors for the zero hot halo outflow reincorporation class.
Code lines: 3
Contained by: file hot_halo.outflow_reincorporation.zero.F90

function: zeroconstructorparameters
Description: Default constructor for the zero hot halo outflow reincorporation class which takes a parameter set as input.
Code lines: 11
Contained by: file hot_halo.outflow_reincorporation.zero.F90
Modules used: input_parameters

function: zerorate
Description: Return the rate of mass reincorporation for outflowed gas in the hot halo.
Code lines: 9
Contained by: file hot_halo.outflow_reincorporation.zero.F90

file: hot_halo.ram_pressure_force.F90
Description: Contains a module that implements a class which provides calculations of ram pressure force.
Code lines: 40

module: hot_halo_ram_pressure_forces
Description: Implements a class which provides calculations of ram pressure force.
Code lines: 18
Contained by: file hot_halo.ram_pressure_force.F90
Modules used: galacticus_nodes
Used by:
- subroutine galacticus_function_classes_destroy
- subroutine galacticus_state_retrieve
- subroutine galacticus_state_store
- file hot_halo.ram_pressure_stripping.Font2008.F90
- file hot_halo.ram_pressure_stripping.timescale.ram_pressure_acceleration.F90
- file ram_pressure_stripping.mass_loss_rate.disks.simple.F90

file ram_pressure_stripping.mass_loss_-
rate.spheroids.simple.F90

file: hot_halo.ram_pressure_force.Font2008.F90
Description: Implements a model of ram pressure stripping of hot halos based on the methods of Font et al. [2008].
Code lines: 103
Modules used: hot_halo_mass_distributions

function: font2008constructorinternal
Description: Internal constructor for the font2008 hot halo ram pressure force class.
Code lines: 9
Contained by: file hot_halo.ram_pressure_force.Font2008.F90
Modules used: input_parameters

function: font2008constructorparameters
Description: Constructor for the font2008 hot halo ram pressure force class which builds the object from a parameter set.
Code lines: 13
Contained by: file hot_halo.ram_pressure_force.Font2008.F90
Modules used: input_parameters

subroutine: font2008destructor
Description: Destructor for the font2008 hot halo ram pressure force class.
Code lines: 7
Contained by: file hot_halo.ram_pressure_force.Font2008.F90

function: font2008force
Description: Return a ram pressure force due to the hot halo using the model of Font et al. [2008].
Code lines: 24
Contained by: file hot_halo.ram_pressure_force.Font2008.F90
Modules used: galacticus_nodes kepler_orbits satellite_orbits

interface: hothalorampressureforcefont2008
Description: Constructors for the font2008 hot halo ram pressure force class.
Code lines: 4
Contained by: file hot_halo.ram_pressure_force.Font2008.F90

file: hot_halo.ram_pressure_force.zero.F90
Description: Implements a model of ram pressure stripping of hot halos which always returns zero force.
Code lines: 60

interface: hothalorampressureforcezero
Description: Constructors for the zero hot halo ram pressure force class.
Code lines: 3
Contained by: file hot_halo.ram_pressure_force.zero.F90

function: zeroconstructorparameters

Description: Constructor for the `zero` hot halo ram pressure force class which builds the object from a parameter set.
Code lines: 10
Contained by: file hot_halo.ram_pressure_force.zero.F90
Modules used: input_parameters

function: `zeroforce`
Description: Return a zero ram pressure force due to the hot halo.
Code lines: 9
Contained by: file hot_halo.ram_pressure_force.zero.F90

file: `hot_halo.ram_pressure_stripping.F90`
Description: Contains a module that implements a class for calculations of ram pressure stripping of hot halos.
Code lines: 41

module: `hot_halo_ram_pressure_stripping`
Description: Implements a class for calculations of ram pressure stripping of hot halos.
Code lines: 19
Contained by: file hot_halo.ram_pressure_stripping.F90
Modules used: galacticus_nodes
Used by: subroutine galacticus_function_classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; subroutine node_component_hot_halo_cold_mode_rate_compute; module node_component_hot_halo_standard

file: `hot_halo.ram_pressure_stripping.Font2008.F90`
Description: Implements a class for ram pressure stripping of hot halos based on the methods of Font et al. [2008].
Code lines: 217
Modules used: dark_matter_halo_scales; hot_halo_mass_distributions; hot_halo_ram_pressure_forces; kind_numbers

function: `font2008constructorinternal`
Description: Internal constructor for the `font2008` hot halo ram pressure stripping class.
Code lines: 12
Contained by: file hot_halo.ram_pressure_stripping.Font2008.F90
Modules used: input_parameters

function: `font2008constructorparameters`
Description: Constructor for the `font2008` hot halo ram pressure stripping class which builds the object from a parameter set.
Code lines: 29
Contained by: file hot_halo.ram_pressure_stripping.Font2008.F90
Modules used: input_parameters

subroutine: `font2008destructor`
Description: Destructor for the `font2008` hot halo ram pressure stripping class.

Code lines: 9
Contained by: file hot_halo.ram_pressure_stripping.Font2008.F90

function: font2008radiussolver
Description: Root function used in finding the ram pressure stripping radius.
Code lines: 19
Contained by: file hot_halo.ram_pressure_stripping.Font2008.F90
Modules used: galactic_structure_enclosed_masses galactic_structure_options numerical_constants_physical

function: font2008radiusstripped
Description: Return the ram pressure stripping radius due to the hot halo using the model of Font et al. [2008].
Code lines: 82
Contained by: file hot_halo.ram_pressure_stripping.Font2008.F90
Modules used: galacticus_display galacticus_error root_finder

interface: hothalorampressurestrippingfont2008
Description: Constructors for the font2008 hot halo ram pressure timescale class.
Code lines: 4
Contained by: file hot_halo.ram_pressure_stripping.Font2008.F90

file: hot_halo.ram_pressure_stripping.timescale.F90
Description: Contains a module that implements a class for calculations of ram pressure stripping timescales for hot halos.
Code lines: 40

module: hot_halo_ram_pressure_stripping_timescales
Description: Implements a class for calculations of ram pressure stripping timescales for hot halos.
Code lines: 18
Contained by: file hot_halo.ram_pressure_stripping.timescale.F90
Modules used: galacticus_nodes
Used by: subroutine galacticus_function_classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; module node_component_hot_halo_standard

file: hot_halo.ram_pressure_stripping.timescale.halo_dynamical_time.F90
Description: Implements a class for the timescale of ram pressure stripping of hot halos in which the timescale is equal to the halo dynamical timescale.
Code lines: 89
Modules used: dark_matter_halo_scales

function: halodynamicaltimeconstructorinternal
Description: Internal constructor for the haloDynamicalTime hot halo ram pressure timescale class.
Code lines: 9
Contained by: file hot_halo.ram_pressure_stripping.timescale.halo_dynamical_time.F90
Modules used: input_parameters

function: `halodynamicaltimeconstructorparameters`
Description: Constructor for the `haloDynamicalTime` hot halo ram pressure timescale class which builds the object from a parameter set.
Code lines: 14
Contained by: file hot_halo.ram_pressure_stripping.timescale.halo_dynamical_time.F90
Modules used: input_parameters

subroutine: `halodynamicaltimedestructor`
Description: Destructor for the `haloDynamicalTime` hot halo ram pressure timescale class.
Code lines: 7
Contained by: file hot_halo.ram_pressure_stripping.timescale.halo_dynamical_time.F90

function: `halodynamicaltimetimescale`
Description: Return a ram pressure timescale due to the hot halo assuming that it equals the halo dynamical time.
Code lines: 8
Contained by: file hot_halo.ram_pressure_stripping.timescale.halo_dynamical_time.F90

interface: `hothalorampressuretimescalehalodynamicaltime`
Description: Constructors for the `haloDynamicalTime` hot halo ram pressure timescale class.
Code lines: 4
Contained by: file hot_halo.ram_pressure_stripping.timescale.halo_dynamical_time.F90

file: `hot_halo.ram_pressure_stripping.timescale.ram_pressure_acceleration.F90`
Description: Implements a class for the timescale of ram pressure stripping of hot halos in which the timescale is estimated from the ram pressure acceleration.
Code lines: 135
Modules used: dark_matter_halo_scales hot_halo_mass_distributions hot_halo_ram_pressure_forces

interface: `hothalorampressuretimescalerampressureacceleration`
Description: Constructors for the `ramPressureAcceleration` hot halo ram pressure timescale class.
Code lines: 4
Contained by: file hot_halo.ram_pressure_stripping.timescale.ram_pressure_acceleration.F90

function: `rampressureaccelerationconstructorinternal`
Description: Internal constructor for the `ramPressureAcceleration` hot halo ram pressure timescale class.
Code lines: 11
Contained by: file hot_halo.ram_pressure_stripping.timescale.ram_pressure_acceleration.F90
Modules used: input_parameters

function: `rampressureaccelerationconstructorparameters`
Description: Constructor for the `ramPressureAcceleration` hot halo ram pressure timescale class which builds the object from a parameter set.
Code lines: 19
Contained by: file hot_halo.ram_pressure_stripping.timescale.ram_pressure_acceleration.F90
Modules used: input_parameters

subroutine: `rampressureaccelerationdestructor`
Description: Destructor for the `ramPressureAcceleration` hot halo ram pressure timescale class.
Code lines: 9
Contained by: file `hot_halo.ram_pressure_stripping.timescale.ram_pressure_acceleration.F90`

function: `rampressureaccelerationtimescale`
Description: Computes the hot halo ram pressure stripping timescale, based on the acceleration due to ram pressure forces. This timescale is approximated as [Roediger and BrÃČÅŠggen, 2007] $\tau \approx \sqrt{2 r_\mathrm{outer} \Sigma_\mathrm{outer} / P_\mathrm{ram}}$, where r_outer is the current outer radius of the hot halo, Σ_outer is the surface density at that radius, and P_ram is the ram pressure force (per unit area). The surface density is approximated as $\Sigma_\mathrm{outer} \approx r_\mathrm{outer} \rho_\mathrm{outer}$, where ρ_outer is the density at the outer radius.
Code lines: 40
Contained by: file `hot_halo.ram_pressure_stripping.timescale.ram_pressure_acceleration.F90`
Modules used: `galacticus_nodes` `numerical_constants_astronomical` `numerical_constants_physical` `numerical_constants_prefixes`

file: `hot_halo.ram_pressure_stripping.virial_radius.F90`
Description: Implements a class for ram pressure stripping which simply returns the virial radius.
Code lines: 86
Modules used: `dark_matter_halo_scales`

interface: `hothalorampressurestrippingvirialradius`
Description: Constructors for the `virialRadius` hot halo ram pressure stripping class.
Code lines: 4
Contained by: file `hot_halo.ram_pressure_stripping.virial_radius.F90`

function: `virialradiusconstructorinternal`
Description: Internal constructor for the `virialRadius` hot halo ram pressure stripping class.
Code lines: 8
Contained by: file `hot_halo.ram_pressure_stripping.virial_radius.F90`

function: `virialradiusconstructorparameters`
Description: Constructor for the `virialRadius` hot halo ram pressure stripping class which builds the object from a parameter set.
Code lines: 13
Contained by: file `hot_halo.ram_pressure_stripping.virial_radius.F90`
Modules used: `input_parameters`

subroutine: `virialradiusdestructor`
Description: Destructor for the `virialRadius` hot halo ram pressure stripping class.
Code lines: 7
Contained by: file `hot_halo.ram_pressure_stripping.virial_radius.F90`

function: `virialradiusradiusstripped`
Description: Return the ram pressure stripping radius which is assumed to be equal to the virial radius.
Code lines: 8
Contained by: file `hot_halo.ram_pressure_stripping.virial_radius.F90`

file: `hot_halo.temperature_profile.Enzo_hydrostatic.F90`
Description: An implementation of the hot halo temperature class which uses the "hydrostatic" solution from the Enzo code.
Code lines: 121
Modules used: `dark_matter_profiles_dmo`

function: `enzohydrostaticconstructorinternal`
Description: Internal constructor for the enzoHydrostatic cooling rate class.
Code lines: 8
Contained by: file `hot_halo.temperature_profile.Enzo_hydrostatic.F90`

function: `enzohydrostaticconstructorparameters`
Description: Constructor for the enzoHydrostatic cooling rate class which builds the object from a parameter set.
Code lines: 13
Contained by: file `hot_halo.temperature_profile.Enzo_hydrostatic.F90`
Modules used: `input_parameters`

subroutine: `enzohydrostaticdestructor`
Description: Destructor for the `enzoHydrostatic` hot halo temperature profile class.
Code lines: 7
Contained by: file `hot_halo.temperature_profile.Enzo_hydrostatic.F90`

function: `enzohydrostatictemperature`
Description: Return the density in a `enzoHydrostatic` hot halo mass distribution.
Code lines: 19
Contained by: file `hot_halo.temperature_profile.Enzo_hydrostatic.F90`
Modules used: `numerical_constants_astronomical` `numerical_constants_atomic` `numerical_constants_physical` `numerical_constants_prefixes`

function: `enzohydrostatictemperaturelogslope`
Description: Return the logarithmic slope of the density profile in a `enzoHydrostatic` hot halo mass distribution.
Code lines: 18
Contained by: file `hot_halo.temperature_profile.Enzo_hydrostatic.F90`
Modules used: `numerical_constants_math`

interface: `hothalotemperatureprofileenzohydrostatic`
Description: Constructors for the enzoHydrostatic hot halo temperature profile class.
Code lines: 4
Contained by: file `hot_halo.temperature_profile.Enzo_hydrostatic.F90`

file: `hot_halo.temperature_profile.F90`
Description: Contains a module which provides a hot halo temperature profile class.
Code lines: 48

module: `hot_halo_temperature_profiles`
Description: Provides a hot halo temperature profile class.

Code lines: 26
Contained by: file hot_halo.temperature_profile.F90
Modules used: galacticus_nodes
Used by: file cooling.cooling_radius.beta_profile.F90; file cooling.cooling_radius.isothermal_profile.F90; file cooling.cooling_radius.simple.F90; subroutine galacticus_function_classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; file hot_halo.mass_distribution.Enzo_hydrostatic.F90; function enzohydrostaticradialmomentintegrand; file nodes.property_extractor.ICM_SZ.F90; file nodes.property_extractor.ICM_Xray_luminosity.F90; module node_component_black_hole_standard

file: hot_halo.temperature_profile.virial.F90
Description: An implementation of the hot halo temperature class which uses an isothermal virial temperature.
Code lines: 103
Modules used: dark_matter_halo_scales

interface: hothalotemperatureprofilevirial
Description: Constructors for the virial hot halo temperature profile class.
Code lines: 4
Contained by: file hot_halo.temperature_profile.virial.F90

function: virialconstructorinternal
Description: Internal constructor for the virial cooling rate class.
Code lines: 8
Contained by: file hot_halo.temperature_profile.virial.F90

function: virialconstructorparameters
Description: Constructor for the virial cooling rate class which builds the object from a parameter set.
Code lines: 13
Contained by: file hot_halo.temperature_profile.virial.F90
Modules used: input_parameters

subroutine: virialdestructor
Description: Destructor for the virial hot halo temperature profile class.
Code lines: 7
Contained by: file hot_halo.temperature_profile.virial.F90

function: virialtemperature
Description: Return the density in a virial hot halo mass distribution.
Code lines: 10
Contained by: file hot_halo.temperature_profile.virial.F90

function: virialtemperaturelogslope
Description: Return the logarithmic slope of the density profile in a virial hot halo mass distribution.

Code lines: 11
Contained by: file hot_halo.temperature_profile.virial.F90

file: inc_gam.F90
Code lines: 1627

module: incomplete_gamma
Code lines: 1624
Contained by: file inc_gam.F90
Modules used: constants_nswc
Used by: function inverse_gamma_function_-incomplete_complementary

function: dgam1
Code lines: 125
Contained by: module incomplete_gamma

function: dgamln
Code lines: 46
Contained by: module incomplete_gamma

function: dgmln1
Code lines: 13
Contained by: module incomplete_gamma

function: dlnrel
Code lines: 34
Contained by: module incomplete_gamma

function: dpdel
Code lines: 29
Contained by: module incomplete_gamma

subroutine: dpni
Code lines: 56
Contained by: module incomplete_gamma

function: drcomp
Code lines: 32
Contained by: module incomplete_gamma

function: drexp
Code lines: 40
Contained by: module incomplete_gamma

function: drlog
Code lines: 61
Contained by: module incomplete_gamma

function: `dsin1`
Code lines: 64
Contained by: module incomplete_gamma

subroutine: `gaminv`
Code lines: 329
Contained by: module incomplete_gamma

subroutine: `gratio`
Code lines: 345
Contained by: module incomplete_gamma

function: `nswc_derf`
Code lines: 39
Contained by: module incomplete_gamma

function: `nswc_derfc0`
Code lines: 63
Contained by: module incomplete_gamma

function: `nswc_derfc1`
Code lines: 64
Contained by: module incomplete_gamma

function: `nswc_derfi`
Code lines: 97
Contained by: module incomplete_gamma

function: `nswc_dgamma`
Code lines: 113
Contained by: module incomplete_gamma

file: `instruments.filters.F90`
Description: Contains a module which implements calculations of filter response curves.
Code lines: 303

module: `instruments_filters`
Description: Implements calculations of filter response curves.
Code lines: 279
Contained by: file instruments.filters.F90
Modules used: `fgsl`, iso_varying_string
Used by: program benchmark_stellar_populations_luminosities, function sedfitconstructorinternal, function lmnstyemssnlineconstructorinternal, function lmnstystllrchrltfll2000constructorinternal, subroutine stellar_luminosities_initialize, subroutine stellar_luminosities_state_restore

subroutine stellar_population_luminosity_tabulate
function filter_luminosity_integrand_ab
function filter_luminosity_integrand
program test_stellar_populations_luminosities

function: filter_extent
Description: Return an array containing the minimum and maximum wavelengths tabulated for this specified filter.
Code lines: 11
Contained by: module instruments_filters

function: filter_get_index
Description: Return the index for the specified filter, loading that filter if necessary.
Code lines: 26
Contained by: module instruments_filters

function: filter_name
Description: Return the name of the specified filter.
Code lines: 8
Contained by: module instruments_filters

function: filter_response
Description: Return the filter response function at the given wavelength (specified in Angstroms). Note that we follow the convention of Hogg et al. [2002] and assume that the filter response gives the fraction of incident photons received by the detector at a given wavelength, multiplied by the relative photon response (which will be 1 for a photon-counting detector such as a CCD, or proportional to the photon energy for a bolometer/calorimeter type detector.
Code lines: 15
Contained by: module instruments_filters
Modules used: numerical_interpolation

subroutine: filter_response_load
Description: Load a filter response curve.
Code lines: 161
Contained by: module instruments_filters
Modules used: file_utilities, fox_dom, galacticus_error, galacticus_hdf5, galacticus_paths, hii_region_emission_lines, io_hdf5, io_xml, memory_management, numerical_constants_units, string_handling

function: filter_vega_offset
Description: Return the Vega to AB magnitude offset for the specified filter.
Code lines: 9
Contained by: module instruments_filters
Modules used: galacticus_error

function: filter_wavelength_effective

Description: Return the effective wavelength for the specified filter.
Code lines: 9
Contained by: module instruments_filters
Modules used: galacticus_error

type: filtertype
Description: A structure which holds filter response curves.
Code lines: 11
Contained by: module instruments_filters

file: interface.CAMB.F90
Description: Contains a module which provides various interfaces to the CAMB code.
Code lines: 511

module: interfaces_camb
Description: Provides various interfaces to the CAMB code.
Code lines: 489
Contained by: file interface.CAMB.F90
Modules used: file_utilities
Used by: subroutine cambcheckrange, subroutine buildtoolcambperform

subroutine: interface_camb_initialize
Description: Initialize the interface with CAMB, including downloading and compiling CAMB if necessary.
Code lines: 60
Contained by: module interfaces_camb
Modules used: file_utilities, galacticus_display, galacticus_error, galacticus_paths, iso_varying_string, system_command

function: flagsretrieve
Description: Retrieve the compiler flags.
Code lines: 10
Contained by: subroutine interface_camb_initialize

subroutine: interface_camb_transfer_function
Description: Run CAMB as necessary to compute transfer functions.
Code lines: 394
Contained by: module interfaces_camb
Modules used: cosmology_parameters, fgsl, file_utilities, galacticus_error, galacticus_paths, hashes_cryptographic, hdf5, input_parameters, io_hdf5, iso_c_binding, iso_varying_string, numerical_constants_astronomical, sort, string_handling, system_command, table_labels, tables

file: `interface.Cloudy.F90`
Description: Contains a module which provides various interfaces to the Cloudy code.
Code lines: 94

module: `interfaces_cloudy`
Description: Provides various interfaces to the Cloudy code.
Code lines: 72
Contained by: file `interface.Cloudy.F90`
Used by: subroutine `interface_cloudy_cie_tabulate` subroutine `buildtoolcloudyperform`

subroutine: `interface_cloudy_initialize`
Description: Initialize the interface with Cloudy, including downloading and compiling Cloudy if necessary.
Code lines: 63
Contained by: module `interfaces_cloudy`
Modules used: `file_utilities` `galacticus_display` `galacticus_error` `galacticus_paths` `iso_varying_string` `system_command`

file: `interface.Cloudy.collisional_ionization_equilibrium.F90`
Description: Contains a module which provides an interface to the Cloudy code for computing tables of cooling functions and chemical state in collisional ionization equilibrium.
Code lines: 233

module: `interfaces_cloudy_cie`
Description: Provides an interface to the Cloudy code for computing tables of cooling functions and chemical state in collisional ionization equilibrium.
Code lines: 211
Contained by: file `interface.Cloudy.collisional_ionization_equilibrium.F90`
Modules used: `file_utilities`
Used by: subroutine `atomiccieclоudytabulate` subroutine `atomiccieclоudytabulate`

subroutine: `interface_cloudy_cie_tabulate`
Description: An interface to the Cloudy code for computing tables of cooling functions and chemical state in collisional ionization equilibrium.
Code lines: 197
Contained by: module `interfaces_cloudy_cie`
Modules used: `galacticus_display` `galacticus_error` `interfaces_cloudy` `io_hdf5` `iso_varying_string` `numerical_ranges` `string_handling` `system_command`

file: `interface.FSPS.F90`
Description: Contains a module which provides various interfaces to the FSPS code [Conroy et al., 2009].
Code lines: 219

module: `interfaces_fsps`

Description: Provides various interfaces to the FSPS code [Conroy et al., 2009].
Code lines: 197
Contained by: file interface.FSPS.F90
Used by: subroutine fspsreadfile, subroutine buildtoolfspsperform

subroutine: interface_fsps_initialize
Description: Initialize the interface with FSPS, including downloading and compiling FSPS if necessary.
Code lines: 70
Contained by: module interfaces_fsps
Modules used: file_utilities, galacticus_display, galacticus_error, galacticus_paths, iso_varying_string, string_handling, system_command

subroutine: interface_fsps_ssps_tabulate
Description: Tabulate simple stellar populations for the given IMF using FSPS.
Code lines: 116
Contained by: module interfaces_fsps
Modules used: dates_and_times, file_utilities, galacticus_error, io_hdf5, iso_varying_string, numerical_constants_astronomical, string_handling, system_command, tables

file: interface.GSL.F90
Description: Contains a module which interfaces with low-level aspects of the GSL library.
Code lines: 77

module: interface_gsl
Description: Interfaces with low-level aspects of the GSL library.
Code lines: 51
Contained by: file interface.GSL.F90
Modules used: iso_c_binding
Used by: module numerical_differentiation

function: gslfunction
Description: Return a c_ptr object for the given function f.
Code lines: 9
Contained by: module interface_gsl
Modules used: iso_c_binding

subroutine: gslfunctiondestroy
Description: Destroy a c_ptr to a gsl_function object.
Code lines: 7
Contained by: module interface_gsl

file: interface.Local_Group_database.F90
Description: Contains a module which interfaces with the Local Group database.
Code lines: 547

module: `interface_local_group_db`
Description: Interfaces with the Local Group database.
Code lines: 525
Contained by: file `interface.Local_Group_database.F90`
Modules used: `fox_dom`
Used by: function subroutine `localgroupdatabaseperform` `localgroupmassfunctionconstructorinternal`

interface: `localgroupdb`
Description: Constructors for the Local Group database class.
Code lines: 3
Contained by: module `interface_local_group_db`

function: `localgroupdbconstructorinternal`
Description: Constructor for the Local Group database class.
Code lines: 14
Contained by: module `interface_local_group_db`
Modules used: `fox_dom` `galacticus_paths` `io_xml` `iso_varying_string`

subroutine: `localgroupdbdestructor`
Description: Destructor for the Local Group database class.
Code lines: 8
Contained by: module `interface_local_group_db`
Modules used: `fox_dom`

subroutine: `localgroupdbgetproperty`
Description: Get a named text property from the Local Group database.
Code lines: 69
Contained by: module `interface_local_group_db`
Modules used: `fox_dom` `galacticus_error` `iso_varying_string`

subroutine: `localgroupdbselect`
Description: Impose a selection on the database.
Code lines: 51
Contained by: module `interface_local_group_db`
Modules used: `fox_dom` `galacticus_error` `iso_varying_string`

subroutine: `localgroupdbselectall`
Description: Select all galaxies in the database.
Code lines: 7
Contained by: module `interface_local_group_db`

subroutine: `localgroupdbupdate`
Description: Update the database.
Code lines: 252

Contained by: module interface_local_group_db
Modules used: fox_dom, galacticus_paths, iso_varying_string, numerical_constants_astronomical

type: vector3d
Description: Vector type.
Code lines: 9
Contained by: module interface_local_group_db

function: vector3dcomparisonunimplemented
Description: Unimplemented comparison operators for 3D vectors.
Code lines: 10
Contained by: module interface_local_group_db
Modules used: galacticus_error

function: vector3dequals
Description: Equality comparison operator for two 3D vectors.
Code lines: 7
Contained by: module interface_local_group_db

file: interface.RecFast.F90
Description: Contains a module which provides various interfaces to the RecFast code.
Code lines: 92

module: interfaces_recfast
Description: Provides various interfaces to the RecFast code.
Code lines: 70
Contained by: file interface.RecFast.F90
Used by: function recfastconstructorinternal, subroutine buildtoolrecfastperform

subroutine: interface_recfast_initialize
Description: Initialize the interface with RecFast, including downloading and compiling RecFast if necessary.
Code lines: 61
Contained by: module interfaces_recfast
Modules used: file_utilities, galacticus_display, galacticus_error, galacticus_paths, iso_varying_string, system_command

file: intergalactic_medium.filtering_mass.F90
Description: Contains a module which provides a class for filtering masses.
Code lines: 39

module: intergalactic_medium_filtering_masses
Description: Provides a class for filtering masses.
Code lines: 17
Contained by: file intergalactic_medium.filtering_mass.F90
Used by: file accretion.halo.Naoz_Barkana_-2007.F90, subroutine galacticus_function_-classes_destroy

subroutine galacticus_state_retrieve
file tasks.intergalactic_medium_-state.F90
function intergalacticmediumstateevolveodes

subroutine galacticus_state_store
function intergalacticmediumstateevolveconstructorinternal

file: intergalactic_medium.filtering_mass.Gnedin2000.F90

Description: Implements the Gnedin [2000] filtering mass calculation.
Code lines: 365
Modules used: cosmology_functions, cosmology_parameters, intergalactic_medium_state, linear_growth, tables

function: gnedin2000coefficientsearlyepoch

Description: Return the coefficients of the fitting function for the filtering mass at early epochs from Naoz and Barkana [2007]. Checks for valid range of redshift and cosmology for the fit to be valid.
Code lines: 23
Contained by: file intergalactic_medium.filtering_mass.Gnedin2000.F90
Modules used: galacticus_error

subroutine: gnedin2000conditionsinitialodes

Description: Compute initial conditions for a system of three variables used to solve for the evolution of the filtering mass. The ODE system to be solved is

$$\dot{y}_1 = y_2 \tag{18.14}$$
$$\dot{y}_2 = -2(\dot{a}/a)D(t)(1+r_{\rm LSS}(t))y_2 f_{\rm DM}{\rm k_B}T(t)/\mu m_{\rm H}a^2 \tag{18.15}$$
$$\dot{y}_3 = 4\pi^4\bar{\rho}(t)\dot{k}_{\rm F}(t)/k_{\rm F}^4(t) \tag{18.16}$$

with initial conditions

$$y_1 = D(t)/k_{\rm F}^2(t) \tag{18.17}$$
$$y_2 = \dot{y}_1 \tag{18.18}$$
$$y_3 = M_{\rm F}(t) \tag{18.19}$$

and where

$$k_{\rm F}(t) = \pi/[M_{\rm F}(t)3/4\pi\bar{\rho}(t)]^{1/3} \tag{18.20}$$

, and $r_{\rm LSS}(t)$ is the function defined by Naoz and Barkana [2007].
Code lines: 48
Contained by: file intergalactic_medium.filtering_mass.Gnedin2000.F90
Modules used: cosmology_parameters, numerical_constants_math

function: gnedin2000constructorinternal

Description: Constructor for the filtering mass class.
Code lines: 12
Contained by: file intergalactic_medium.filtering_mass.Gnedin2000.F90

function: gnedin2000constructorparameters

Description: Default constructor for the file IGM state class.
Code lines: 22
Contained by: file intergalactic_medium.filtering_mass.Gnedin2000.F90
Modules used: input_parameters

subroutine: gnedin2000destructor
Description: Destructor for the filtering mass class.
Code lines: 10
Contained by: file intergalactic_medium.filtering_mass.Gnedin2000.F90

function: gnedin2000massfiltering
Description: Return the filtering mass at the given time.
Code lines: 9
Contained by: file intergalactic_medium.filtering_mass.Gnedin2000.F90

function: gnedin2000massfilteringearlyepoch
Description: Fitting function for the filtering mass at early epochs from Naoz and Barkana [2007]. Checks for valid range of redshift and cosmology for the fit to be valid.
Code lines: 18
Contained by: file intergalactic_medium.filtering_mass.Gnedin2000.F90

function: gnedin2000odes
Description: Compute the rates of change of the filtering mass ODE system.
Code lines: 31
Contained by: file intergalactic_medium.filtering_mass.Gnedin2000.F90
Modules used: numerical_constants_astronomical numerical_constants_math numerical_constants_physical

function: gnedin2000rlss
Description: Evaluate the $r_{\rm LSS}$ parameter of Naoz and Barkana [2007] using their fitting formula.
Code lines: 13
Contained by: file intergalactic_medium.filtering_mass.Gnedin2000.F90

subroutine: gnedin2000tabulate
Description: Construct a table of filtering mass as a function of cosmological time.
Code lines: 70
Contained by: file intergalactic_medium.filtering_mass.Gnedin2000.F90
Modules used: fodeiv2 galacticus_error numerical_constants_math odeiv2_solver

function: massfilteringodes
Description: Evaluates the ODEs controlling the evolution temperature.
Code lines: 20
Contained by: subroutine gnedin2000tabulate
Modules used: fgsl numerical_constants_astronomical numerical_constants_atomic

interface: intergalacticmediumfilteringmassgnedin2000
Description: Constructors for the filtering mass class.

Code lines: 4
Contained by: file intergalactic_medium.filtering_mass.Gnedin2000.F90

file: intergalactic_medium.state.F90
Description: Contains a module which provides a class for calculations of the intergalactic medium thermal and ionization state.
Code lines: 247

module: intergalactic_medium_state
Description: Provides a class for calculations of the intergalactic medium thermal and ionization state.
Code lines: 225
Contained by: file intergalactic_medium.state.F90
Modules used: cosmology_functions, cosmology_parameters, tables
Used by: file accretion.halo.simple.F90; subroutine galacticus_function_classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; file intergalactic_medium.filtering_mass.Gnedin2000.F90; file radiation.intergalactic_background.internal.F90; file tasks.intergalactic_medium_state.F90; file universe.operators.intergalactic_medium_state_evolve.F90

subroutine: intergalacticmediumstateelectronscatteringtabulate
Description: Construct a table of electron scattering optical depth as a function of cosmological time.
Code lines: 61
Contained by: module intergalactic_medium_state
Modules used: fgsl, numerical_integration

function: intergalacticmediumstateelectronscatteringintegrand
Description: Integrand for electron scattering optical depth calculations.
Code lines: 16
Contained by: subroutine intergalacticmediumstateelectronscatteringtabulate
Modules used: numerical_constants_astronomical, numerical_constants_physical

file: intergalactic_medium.state.RecFast.F90
Description: An implementation of the intergalactic medium state class in which state is computed using RecFast.
Code lines: 183
Modules used: file_utilities

interface: intergalacticmediumstaterecfast
Description: Constructors for the RecFast intergalactic medium state class.
Code lines: 4
Contained by: file intergalactic_medium.state.RecFast.F90

function: recfastconstructorinternal
Description: Constructor for the RecFast IGM state class.
Code lines: 121
Contained by: file intergalactic_medium.state.RecFast.F90

Modules used: cosmology_parameters, dates_and_times, file_utilities, galacticus_display, galacticus_error, galacticus_paths, interfaces_recfast, io_hdf5, numerical_constants_astronomical, system_command

function: recfastconstructorparameters
Description: Default constructor for the RecFast IGM state class.
Code lines: 17
Contained by: file intergalactic_medium.state.RecFast.F90
Modules used: input_parameters

file: intergalactic_medium.state.file.F90
Description: An implementation of the intergalactic medium state class in which state is read from file.
Code lines: 229
Modules used: fgsl

function: fileconstructorinternal
Description: Constructor for the file IGM state class.
Code lines: 12
Contained by: file intergalactic_medium.state.file.F90
Modules used: file_utilities

function: fileconstructorparameters
Description: Default constructor for the file IGM state class.
Code lines: 25
Contained by: file intergalactic_medium.state.file.F90
Modules used: input_parameters

subroutine: filedestructor
Description: Destructor for the file IGM state class.
Code lines: 8
Contained by: file intergalactic_medium.state.file.F90

function: fileelectronfraction
Description: Return the electron fraction in the intergalactic medium at the specified time by interpolating in tabulated data,
Code lines: 12
Contained by: file intergalactic_medium.state.file.F90
Modules used: numerical_interpolation

function: fileneutralheliumfraction
Description: Return the neutral helium fraction in the intergalactic medium at the specified time by interpolating in tabulated data,
Code lines: 12
Contained by: file intergalactic_medium.state.file.F90
Modules used: numerical_interpolation

function: fileneutralhydrogenfraction
Description: Return the neutral hydrogen fraction in the intergalactic medium at the specified time by interpolating in tabulated data,

Code lines: 12
Contained by: file intergalactic_medium.state.file.F90
Modules used: numerical_interpolation

subroutine: filereaddata
Description: Read in data describing the state of the intergalactic medium.
Code lines: 44
Contained by: file intergalactic_medium.state.file.F90
Modules used: file_utilities galacticus_error io_hdf5 table_labels

function: filesinglyionizedheliumfraction
Description: Return the neutral helium fraction in the intergalactic medium at the specified time by interpolating in tabulated data,
Code lines: 12
Contained by: file intergalactic_medium.state.file.F90
Modules used: numerical_interpolation

function: filetemperature
Description: Return the temperature of the intergalactic medium at the specified time by interpolating in tabulated data.
Code lines: 12
Contained by: file intergalactic_medium.state.file.F90
Modules used: numerical_interpolation

interface: intergalacticmediumstatefile
Description: Constructors for the file intergalactic medium state class.
Code lines: 4
Contained by: file intergalactic_medium.state.file.F90

file: intergalactic_medium.state.instant_reionization.F90
Description: An implementation of the intergalactic medium state class for a simplistic model of instantaneous and full reionization with some other class providing the pre-reionization state.
Code lines: 271

subroutine: instantreionizationdestructor
Description: Destructor for the instant reionization intergalactic medium state class.
Code lines: 9
Contained by: file intergalactic_medium.state.instant_reionization.F90

function: instantreionizationelectronfraction
Description: Return the electron fraction of the IGM in the instantReionization model.
Code lines: 13
Contained by: file intergalactic_medium.state.instant_reionization.F90
Modules used: numerical_constants_astronomical

function: instantreionizationigmconstructorinternal
Description: Constructor for the instantReionization IGM state class.
Code lines: 63
Contained by: file intergalactic_medium.state.instant_reionization.F90

Modules used: galacticus_error root_finder

function: opticaldepth
Description: Compute the optical depth to electron scattering assuming instant reionization at the given time. Subtract from this the target optical depth so that we have a function whose root gives the epoch of instantaneous reionization.
Code lines: 12
Contained by: function instantreionizationigmconstructorinternal

function: instantreionizationigmconstructorparameters
Description: Constructor for the instantReionization IGM state class which takes a parameter set as input.
Code lines: 70
Contained by: file intergalactic_medium.state.instant_reionization.F90
Modules used: input_parameters

function: instantreionizationneutralheliumfraction
Description: Return the neutral helium fraction of the IGM in the instantReionization model.
Code lines: 13
Contained by: file intergalactic_medium.state.instant_reionization.F90
Modules used: numerical_constants_astronomical

function: instantreionizationneutralhydrogenfraction
Description: Return the neutral hydrogen fraction of the IGM in the instantReionization model.
Code lines: 13
Contained by: file intergalactic_medium.state.instant_reionization.F90
Modules used: numerical_constants_astronomical

function: instantreionizationsinglyionizedheliumfraction
Description: Return the singly-ionized helium fraction of the IGM in the instantReionization model.
Code lines: 13
Contained by: file intergalactic_medium.state.instant_reionization.F90
Modules used: numerical_constants_astronomical

function: instantreionizationtemperature
Description: Return the temperature of the IGM in the instantReionization model.
Code lines: 15
Contained by: file intergalactic_medium.state.instant_reionization.F90

interface: intergalacticmediumstateinstantreionization
Description: Constructors for the instantReionization intergalactic medium state class.
Code lines: 4
Contained by: file intergalactic_medium.state.instant_reionization.F90

file: intergalactic_medium.state.internal.F90
Description: An implementation of the intergalactic medium state class for an internal model of instantaneous and full reionization.
Code lines: 334

interface: intergalacticmediumstateinternal

Description: Constructors for the internal intergalactic medium state class.
Code lines: 4
Contained by: file intergalactic_medium.state.internal.F90

subroutine: `internalautohook`
Description: Hook into the internal intergalactic medium state evolver to receive updates.
Code lines: 8
Contained by: file intergalactic_medium.state.internal.F90
Modules used: events_hooks

function: `internalconstructorinternal`
Description: Internal constructor for the `internal` IGM state class.
Code lines: 25
Contained by: file intergalactic_medium.state.internal.F90

function: `internalconstructorparameters`
Description: Constructor for the `internal` IGM state class which takes a parameter set as input.
Code lines: 16
Contained by: file intergalactic_medium.state.internal.F90
Modules used: input_parameters

subroutine: `internaldestructor`
Description: Destructor for the internal IGM state class.
Code lines: 8
Contained by: file intergalactic_medium.state.internal.F90

function: `internalelectronfraction`
Description: Return the electron fraction of the IGM in the internal model.
Code lines: 37
Contained by: file intergalactic_medium.state.internal.F90
Modules used: fgsl iso_c_binding numerical_interpolation

function: `internalneutralheliumfraction`
Description: Return the neutral helium fraction of the IGM in the internal model.
Code lines: 35
Contained by: file intergalactic_medium.state.internal.F90
Modules used: fgsl iso_c_binding numerical_interpolation

function: `internalneutralhydrogenfraction`
Description: Return the neutral hydrogen fraction of the IGM in the internal model.
Code lines: 35
Contained by: file intergalactic_medium.state.internal.F90
Modules used: fgsl iso_c_binding numerical_interpolation

function: `internalsinglyionizedheliumfraction`
Description: Return the singly ionized helium fraction of the IGM in the internal model.
Code lines: 35

Contained by: file intergalactic_medium.state.internal.F90
Modules used: fgsl iso_c_binding numerical_interpolation

subroutine: internalstateset
Description: Set state in the internal intergalatic medium state class.
Code lines: 38
Contained by: file intergalactic_medium.state.internal.F90
Modules used: galacticus_error memory_management

function: internaltemperature
Description: Return the temperature of the IGM in the internal model.
Code lines: 31
Contained by: file intergalactic_medium.state.internal.F90
Modules used: fgsl iso_c_binding numerical_interpolation

file: intergalactic_medium.state.simple.F90
Description: An implementation of the intergalactic medium state class for a simplistic model of instantaneous and full reionization.
Code lines: 191

interface: intergalacticmediumstatesimple
Description: Constructors for the simple intergalactic medium state class.
Code lines: 4
Contained by: file intergalactic_medium.state.simple.F90

subroutine: simpledestructor
Description: Destructor for the simple IGM state class.
Code lines: 8
Contained by: file intergalactic_medium.state.simple.F90

function: simpleelectronfraction
Description: Return the electron fraction of the IGM in the simple model.
Code lines: 13
Contained by: file intergalactic_medium.state.simple.F90
Modules used: numerical_constants_astronomical

function: simpleigmconstructorinternal
Description: Constructor for the simple IGM state class.
Code lines: 11
Contained by: file intergalactic_medium.state.simple.F90

function: simpleigmconstructorparameters
Description: Constructor for the simple IGM state class which takes a parameter set as input.
Code lines: 47
Contained by: file intergalactic_medium.state.simple.F90
Modules used: input_parameters

function: `simpleneutralheliumfraction`
Description: Return the neutral helium fraction of the IGM in the simple model.
Code lines: 13
Contained by: file `intergalactic_medium.state.simple.F90`
Modules used: `numerical_constants_astronomical`

function: `simpleneutralhydrogenfraction`
Description: Return the neutral hydrogen fraction of the IGM in the simple model.
Code lines: 13
Contained by: file `intergalactic_medium.state.simple.F90`
Modules used: `numerical_constants_astronomical`

function: `simplesinglyionizedheliumfraction`
Description: Return the singly-ionized helium fraction of the IGM in the simple model.
Code lines: 13
Contained by: file `intergalactic_medium.state.simple.F90`
Modules used: `numerical_constants_astronomical`

function: `simpletemperature`
Description: Return the temperature of the IGM in the simple model.
Code lines: 12
Contained by: file `intergalactic_medium.state.simple.F90`

file: `iso_varying_string.F90`
Code lines: 2731

module: `iso_varying_string`
Code lines: 2690
Contained by: file `iso_varying_string.F90`
Used by:

- program `galacticus`
- program `benchmark_stellar_populations_luminosities`
- subroutine `ciefilereadfile`
- function `isothermalconstructorinternal`
- function `matterlambdaconstructorinternal`
- subroutine `assertpropertiesgettable`
- subroutine `inter_tree_event_post_evolve`
- function `node_push_from_tree`
- subroutine `galacticus_state_store_null`
- function `galactic_structure_radius_enclosing_mass`
- subroutine `radiussolve`
- module `galacticus_error`
- subroutine `galacticus_meta_evolver_profile`
- subroutine `atomic_data_initialize`
- subroutine `ciefilereadfile`
- function `betaprofileconstructorinternal`
- function `simpleconstructorinternal`
- function `matterlambdaexpansionfactor`
- function `node_branch_jump`
- function `node_pull_from_tree`
- function `node_subhalo_promotion`
- function `galactic_structure_radius_enclosing_density`
- module `galactic_structure_radii_definitions`
- module `galacticus_display`
- module `galacticus_paths`
- subroutine `galacticus_meta_evolver_profiler_output`

module galacticus_output_open
function colordistributionsdssconstructorinternal
function concentrationvshalomasscdmludlow2016constructorinternal
function galaxysizessdssconstructorinternal
function luminosityfunctionhalphaconstructorinternal
function massfunctionstellarconstructorinternal
function massfunctionstellarukidssudsconstructorinternal
function massfunctionstellarvipersconstructorinternal
file galacticus.output.analyses.mean_function_1d.F90
function spindistributionbett2007constructorinternal
function output_analysis_output_weight_survey_volume
subroutine galacticus_build_output
subroutine galacticus_extra_output_halo_fourier_profile
subroutine galacticus_version_output
module galacticus_state
function squareisinlightcone
function geometrymangleangularpower
function geometrymanglesolidangle
subroutine caputi2011ukidssudsrandomsinitialize
subroutine martin2010alfalfarandomsinitialize
module hii_region_emission_lines
subroutine interface_camb_initialize
subroutine interface_cloudy_initialize
subroutine interface_fsps_initialize
function localgroupdbconstructorinternal

function hivshalomassrelationpadmanabhan2017constructorinternal
function concentrationdistributioncdmcococonstructorinternal
function diskstructurenfwconstructorinternal
function luminosityfunctionconstructorinternal
function massfunctionhiconstructorinternal
function massfunctionstellarprimusconstructorinternal
function massfunctionstellarultravistaconstructorinternal
function massfunctionstellarzfourgeconstructorinternal
file galacticus.output.analyses.scatter_function_1d.F90
function stellarvshalomassrelationleauthaud2012constructorinternal
file galacticus.output.analyses.volume_function_1d.F90
function galacticus_build_string
subroutine galacticus_version
function galacticus_version_string
function squareconstructorinternal
function squareposition
subroutine geometrymanglebuild
subroutine windowread
subroutine liwhite2009sdssrandomsinitialize
subroutine fullskywindowfunctions
module instruments_filters
subroutine interface_camb_transfer_function
subroutine interface_cloudy_cie_tabulate
subroutine interface_fsps_ssps_tabulate
subroutine localgroupdbgetproperty

subroutine localgroupdbselect
subroutine localgroupdbupdate
subroutine interface_recfast_initialize
function inverse_gamma_function_-incomplete_complementary
function generalizedpressschechterfractionsubresolutionmass
function generalizedpressschechtermassbranch
subroutine cole2000build
subroutine readrootnodeaffinitiesinitial
subroutine readscanforbranchjumps
module merger_tree_read_importers
file merger_-trees.construct.read.importer.SussingMergerTrees.F90
function nodearrayposition
module merger_trees_dump_evolution
module merger_tree_dump_structure
function historytimeevolveto
subroutine recordevolutionoutput
function recordevolutiontimeevolveto
subroutine evolve_to_time_report
subroutine satellitemergerprocess
function satellitetimeevolveto
function simpletimeevolveto
function standardtimeevolveto
subroutine augmentfinalize
subroutine conditionalmffinalize
subroutine massaccretionhistoryoperate
subroutine outputrootmassesfinalize
file merger_-trees.operators.particulate.F90
subroutine particulateoperate
subroutine merger_tree_structure_output
function sedfitconstructorinternal
function independentlikelihoodssequantialevaluate
function multivariatenormalstochasticconstructorparameters
function posteriorasprioriconstructorinternal
file nBody.import.GadgetBinary.F90
subroutine selfboundoperate
file nodes.property_-extractor.luminosity_emission_-line.F90
file nodes.property_-extractor.luminosity_stellar.F90
file nodes.property_-extractor.luminosity_-stellar.dust.CharlotFall2000.F90
subroutine odeiv2_solve
function multivectorizedcompositetrapezoidal1devaluate
function interpolate
function interpolate_derivative
function root_finder_find
function search_array_varstring
module sort
module abundances_structure
module chemical_abundances_structure
module chemical_structures
module function_classes
subroutine history_dump
subroutine history_extend
subroutine history_long_integer_dump
subroutine kepler_orbits_dump
module merger_tree_data_structure
subroutine merger_tree_dump
module galacticus_nodes
function agestatisticsdiskintegratedsfrattributematch
function agestatisticsdisktimeweightedintegratedsfrattributematch
function agestatisticsspheroidintegratedsfrattributematch
function agestatisticsspheroidtimeweightedintegratedsfrattributematch

function basicaccretionrateattributematch
function basicaccretionratebertschingerattributematch
function basicmassattributematch
function basicmassbertschingerattributematch
function basicmassmaximumattributematch
function basicmasstargetattributematch
function basicradiusturnaroundattributematch
function basicradiusturnaroundgrowthrateattributematch
function basictimeattributematch
function basictimelastisolatedattributematch
function blackholeaccretionrateattributematch
function blackholemassattributematch
function blackholemassseedattributematch
function blackholeradialpositionattributematch
function blackholeradiativeefficiencyattributematch
function blackholespinattributematch
function blackholespinseedattributematch
function blackholetripleinteractiontimeattributematch
function darkmatterprofilescaleattributematch
function darkmatterprofilescalegrowthrateattributematch
function darkmatterprofilescaleislimitedattributematch
function darkmatterprofileshapeattributematch
function darkmatterprofileshapegrowthrateattributematch
function diskabundancesgasattributematch
function diskabundancesstellarattributematch
function diskangularmomentumattributematch
function diskfractionmassretainedattributematch
function diskhalfmassradiusattributematch
function diskisinitializedattributematch
function diskluminositiesstellarattributematch
function diskmassgasattributematch
function diskmassstellarattributematch
function diskmassstellarformedattributematch
function diskradiusattributematch
function diskstarformationhistoryattributematch
function diskstarformationrateattributematch
function diskstellarpropertieshistoryattributematch
function diskvelocityattributematch
function dynamicsstatisticsadiabaticratioattributematch
function dynamicsstatisticsbarinstabilitytimescaleattributematch
function dynamicsstatisticstimeattributematch
function formationtimeformationtimeattributematch
function hosthistoryhostmassmaximumattributematch
function hothaloabundancesattributematch
function hothaloabundancescoldattributematch
function hothaloangularmomentumattributematch

function hothaloangularmomentumcoldattributematch
function hothalochemicalsattributematch
function hothaloheatsourceattributematch
function hothalohothalocoolingabundancesattributematch
function hothalohothalocoolingangularmomentumattributematch
function hothalohothalocoolingmassattributematch
function hothaloisinitializedattributematch
function hothalomassattributematch
function hothalomasscoldattributematch
function hothalomasssinkattributematch
function hothalomasstotalattributematch
function hothaloouterradiusattributematch
function hothalooutflowedabundancesattributematch
function hothalooutflowedangularmomentumattributematch
function hothalooutflowedmassattributematch
function hothalooutflowingabundancesattributematch
function hothalooutflowingangularmomentumattributematch
function hothalooutflowingmassattributematch
function hothalostrippedabundancesattributematch
function hothalostrippedmassattributematch
function hothalotrackedoutflowabundancesattributematch
function hothalotrackedoutflowmassattributematch
function hothalounaccretedmassattributematch
function indicesbranchtipattributematch
function interoutputdiskstarformationrateattributematch
function interoutputspheroidstarformationrateattributematch
function massflowstatisticscooledmassattributematch
function mergingstatisticsgalaxymajormergertimeattributematch
function mergingstatisticsmajormergertimeattributematch
function mergingstatisticsmasswhenfirstisolatedattributematch
function mergingstatisticsnodeformationtimeattributematch
function mergingstatisticsnodehierarchylevelattributematch
function mergingstatisticsnodehierarchylevelmaximumattributematch
function mergingstatisticsnodemajormergertimeattributematch
function mergingstatisticsrecentmajormergercountattributematch
function nbodyintegersattributematch
function nbodyrealsattributematch
subroutine nodeclasshierarchyinitialize
subroutine nodecomponentagestatisticscreate
subroutine nodecomponentagestatisticsdumpascii
function nodecomponentagestatisticsnullnamefromindex
subroutine nodecomponentagestatisticsnullserializeascii
subroutine nodecomponentagestatisticsserializeascii
function nodecomponentagestatisticsstandardnamefromindex
subroutine nodecomponentagestatisticsstandardserializeascii
subroutine nodecomponentbasiccreate

subroutine nodecomponentbasicdumpascii
function nodecomponentbasicextendedtrackingnamefromindex
subroutine nodecomponentbasicextendedtrackingserializeascii
function nodecomponentbasicnonevolvingnamefromindex
subroutine nodecomponentbasicnonevolvingserializeascii
function nodecomponentbasicnullnamefromindex
subroutine nodecomponentbasicnullserializeascii
subroutine nodecomponentbasicserializeascii
function nodecomponentbasicstandardextendednamefromindex
subroutine nodecomponentbasicstandardextendedserializeascii
function nodecomponentbasicstandardnamefromindex
subroutine nodecomponentbasicstandardserializeascii
function nodecomponentbasicstandardtrackingnamefromindex
subroutine nodecomponentbasicstandardtrackingserializeascii
subroutine nodecomponentblackholecreate
subroutine nodecomponentblackholedumpascii
function nodecomponentblackholenoncentralnamefromindex
subroutine nodecomponentblackholenoncentralserializeascii
function nodecomponentblackholenullnamefromindex
subroutine nodecomponentblackholenullserializeascii
subroutine nodecomponentblackholeserializeascii
function nodecomponentblackholesimplenamefromindex
subroutine nodecomponentblackholesimpleserializeascii
function nodecomponentblackholestandardnamefromindex
subroutine nodecomponentblackholestandardserializeascii
subroutine nodecomponentdarkmatterprofilecreate
subroutine nodecomponentdarkmatterprofiledumpascii
function nodecomponentdarkmatterprofilenullnamefromindex
subroutine nodecomponentdarkmatterprofilenullserializeascii
function nodecomponentdarkmatterprofilescalenamefromindex
function nodecomponentdarkmatterprofilescalepresetnamefromindex
subroutine nodecomponentdarkmatterprofilescalepresetserializeascii
subroutine nodecomponentdarkmatterprofilescaleserializeascii
function nodecomponentdarkmatterprofilescaleshapenamefromindex
subroutine nodecomponentdarkmatterprofilescaleshapeserializeascii
subroutine nodecomponentdarkmatterprofileserializeascii
subroutine nodecomponentdiskcreate
subroutine nodecomponentdiskdumpascii
function nodecomponentdisknullnamefromindex
subroutine nodecomponentdisknullserializeascii
subroutine nodecomponentdiskserializeascii
function nodecomponentdiskstandardnamefromindex
subroutine nodecomponentdiskstandardserializeascii
function nodecomponentdiskverysimplenamefromindex
subroutine nodecomponentdiskverysimpleserializeascii
function nodecomponentdiskverysimplesizenamefromindex

subroutine nodecomponentdiskverysimplesizeserializeascii
subroutine nodecomponentdynamicsstatisticsbarsserialize
subroutine nodecomponentdynamicsstatisticsdumpascii
subroutine nodecomponentdynamicsstatisticsnullserialize
function nodecomponentformationtimecole2000namefromindex
subroutine nodecomponentformationtimecreate
function nodecomponentformationtimemassfractionnamefromindex
function nodecomponentformationtimenullnamefromindex
subroutine nodecomponentformationtimeserializeascii
subroutine nodecomponenthosthistorycreate
function nodecomponenthosthistorynullnamefromindex
subroutine nodecomponenthosthistoryserializeascii
subroutine nodecomponenthosthistorystandardserialize
subroutine nodecomponenthothalocoldmodeserializeascii
subroutine nodecomponenthothalodumpascii
subroutine nodecomponenthothalonullserializeascii
subroutine nodecomponenthothalooutflowtrackingserialize
function nodecomponenthothalostandardnamefromindex
function nodecomponenthothaloverysimpledelayednamefromindex
function nodecomponenthothaloverysimplenamefromindex
subroutine nodecomponentindicescreate
function nodecomponentindicesnullnamefromindex
function nodecomponentdynamicsstatisticsbarsnamefromindex
subroutine nodecomponentdynamicsstatisticscreate
function nodecomponentdynamicsstatisticsnullnamefromindex
subroutine nodecomponentdynamicsstatisticsserializeascii
subroutine nodecomponentformationtimecole2000serializeascii
subroutine nodecomponentformationtimedumpascii
subroutine nodecomponentformationtimemassfractionserializeascii
subroutine nodecomponentformationtimenullserializeascii
subroutine nodecomponentgeterror
subroutine nodecomponenthosthistorydumpascii
subroutine nodecomponenthosthistorynullserializeascii
function nodecomponenthosthistorystandardnamefromindex
function nodecomponenthothalocoldmodenamefromindex
subroutine nodecomponenthothalocreate
function nodecomponenthothalonullnamefromindex
function nodecomponenthothalooutflowtrackingnamefromindex
subroutine nodecomponenthothaloserializeascii
subroutine nodecomponenthothalostandardserializeascii
subroutine nodecomponenthothaloverysimpledelayedserializeascii
subroutine nodecomponenthothaloverysimpleserializeascii
subroutine nodecomponentindicesdumpascii
subroutine nodecomponentindicesnullserializeascii

subroutine nodecomponentindicesserializeascii
subroutine nodecomponentindicesstandardserializeascii
subroutine nodecomponentinteroutputdumpascii
subroutine nodecomponentinteroutputnullserializeascii
function nodecomponentinteroutputstandardnamefromindex
subroutine nodecomponentmassflowstatisticscreate
function nodecomponentmassflowstatisticsnullnamefromindex
subroutine nodecomponentmassflowstatisticsserializeascii
subroutine nodecomponentmassflowstatisticsstandardserializeascii
subroutine nodecomponentmergingstatisticsdumpascii
subroutine nodecomponentmergingstatisticsmajorserializeascii
subroutine nodecomponentmergingstatisticsnullserializeascii
subroutine nodecomponentmergingstatisticsrecentserializeascii
function nodecomponentmergingstatisticsstandardnamefromindex
function nodecomponentnamefromindex
subroutine nodecomponentnbodydumpascii
subroutine nodecomponentnbodygenericserializeascii
subroutine nodecomponentnbodynullserializeascii
subroutine nodecomponentpositioncreate
function nodecomponentpositionnullnamefromindex
function nodecomponentpositionpresetnamefromindex
subroutine nodecomponentpositionpresetorphansserializeascii
subroutine nodecomponentpositionserializeascii
function nodecomponentindicesstandardnamefromindex
subroutine nodecomponentinteroutputcreate
function nodecomponentinteroutputnullnamefromindex
subroutine nodecomponentinteroutputserializeascii
subroutine nodecomponentinteroutputstandardserializeascii
subroutine nodecomponentmassflowstatisticsdumpascii
subroutine nodecomponentmassflowstatisticsnullserializeascii
function nodecomponentmassflowstatisticsstandardnamefromindex
subroutine nodecomponentmergingstatisticscreate
function nodecomponentmergingstatisticsmajornamefromindex
function nodecomponentmergingstatisticsnullnamefromindex
function nodecomponentmergingstatisticsrecentnamefromindex
subroutine nodecomponentmergingstatisticsserializeascii
subroutine nodecomponentmergingstatisticsstandardserializeascii
subroutine nodecomponentnbodycreate
function nodecomponentnbodygenericnamefromindex
function nodecomponentnbodynullnamefromindex
subroutine nodecomponentnbodyserializeascii
subroutine nodecomponentpositiondumpascii
subroutine nodecomponentpositionnullserializeascii
function nodecomponentpositionpresetorphansnamefromindex
subroutine nodecomponentpositionpresetserializeascii
function nodecomponentpositiontracedarkmatternamefromindex

subroutine nodecomponentpositiontracedarkmatterserializeascii
subroutine nodecomponentsatellitecreate
subroutine nodecomponentsatellitedumpascii
function nodecomponentsatellitenullnamefromindex
subroutine nodecomponentsatellitenullserializeascii
function nodecomponentsatelliteorbitingnamefromindex
subroutine nodecomponentsatelliteorbitingserializeascii
function nodecomponentsatellitepresetnamefromindex
subroutine nodecomponentsatellitepresetserializeascii
subroutine nodecomponentsatelliteserializeascii
function nodecomponentsatellitestandardnamefromindex
subroutine nodecomponentsatellitestandardserializeascii
function nodecomponentsatelliteverysimplenamefromindex
subroutine nodecomponentsatelliteverysimpleserializeascii
subroutine nodecomponentspheroidcreate
subroutine nodecomponentspheroiddumpascii
function nodecomponentspheroidnullnamefromindex
subroutine nodecomponentspheroidnullserializeascii
subroutine nodecomponentspheroidserializeascii
function nodecomponentspheroidstandardnamefromindex
subroutine nodecomponentspheroidstandardserializeascii
function nodecomponentspheroidverysimplenamefromindex
subroutine nodecomponentspheroidverysimpleserializeascii
subroutine nodecomponentspincreate
subroutine nodecomponentspindumpascii
function nodecomponentspinnullnamefromindex
subroutine nodecomponentspinnullserializeascii
function nodecomponentspinpreset3dnamefromindex
subroutine nodecomponentspinpreset3dserializeascii
function nodecomponentspinpresetnamefromindex
subroutine nodecomponentspinpresetserializeascii
function nodecomponentspinrandomnamefromindex
subroutine nodecomponentspinrandomserializeascii
subroutine nodecomponentspinserializeascii
function nodecomponentspinvitvitskanamefromindex
subroutine nodecomponentspinvitvitskaserializeascii
function positionpositionattributematch
function positionpositionhistoryattributematch
function positionpositionorphanattributematch
function positiontimeassignattributematch
function positionvelocityattributematch
function positionvelocityorphanattributematch
function satelliteboundmassattributematch
function satelliteboundmasshistoryattributematch

function satelliteisorphanattributematch
function satellitemergetimeattributematch
function satellitenodeindexattributematch
function satellitenodeindexhistoryattributematch
function satellitepositionattributematch
function satellitetidalheatingnormalizedattributematch
function satellitetidaltensorpathintegratedattributematch
function satellitetimeofmergingattributematch
function satellitevelocityattributematch
function satellitevirialorbitattributematch
function spheroidabundancesgasattributematch
function spheroidabundancesstellarattributematch
function spheroidangularmomentumattributematch
function spheroidenergygasinputattributematch
function spheroidhalfmassradiusattributematch
function spheroidisinitializedattributematch
function spheroidluminositiesstellarattributematch
function spheroidmassgasattributematch
function spheroidmassgassinkattributematch
function spheroidmassstellarattributematch
function spheroidmassstellarformedattributematch
function spheroidradiusattributematch
function spheroidstarformationhistoryattributematch
function spheroidstarformationrateattributematch
function spheroidstellarpropertieshistoryattributematch
function spheroidvelocityattributematch
function spinspinattributematch
function spinspingrowthrateattributematch
function spinspinvectorattributematch
function spinspinvectorgrowthrateattributematch
function treenodepropertynamefromindex
subroutine treenodeserializeascii
subroutine treenodeserializexml
subroutine node_component_basic_-extended_bindings
subroutine node_component_basic_-standard_promote
subroutine node_component_black_hole_-standard_output_properties
module node_component_disk_standard
subroutine node_component_disk_-standard_post_step
module node_component_disk_very_simple
module node_component_disk_very_-simple_size
subroutine node_component_dynamics_-statistics_bars_output
subroutine node_component_formation_-time_mass_fraction_node_promotion
module node_component_hot_halo_cold_-mode
subroutine node_component_hot_halo_-standard_initialize
subroutine node_component_merging_-statistics_major_output
subroutine node_component_merging_-statistics_recent_initialize

module node_component_nbody_generic

subroutine node_component_satellite_-preset_inter_tree_postprocess
subroutine node_component_satellite_-preset_satellite_host_change
subroutine node_component_spheroid_-standard_post_step
module node_component_spin_vitvitska

module stellar_luminosities_structure

subroutine stellar_luminosities_state_-restore
module tensors
subroutine tensor_r2_d3_sym_dump_raw
file posterior_-sampling.convergence.Gelman-Rubin.F90
function gelmanrubinisconverged
function adaptiveexponent

subroutine tempereddifferentialevolutionupdate
subroutine correlationrestore
file posterior_-sampling.state.initialize.resume.F90
function intergalacticbackgroundinternalupdate
subroutine satellite_move_to_new_host

module stellar_astrophysics
function hegerwoosley2002constructorinternal
module stellar_population_luminosities

subroutine excursion_sets_barriers_-remap_null_initialize
function rodriguezpuebla2016constructorinternal
function cosmicemuvalue

subroutine restore_table

subroutine spherical_collapse_matter_-lambda_critical_overdensity_tabulate

subroutine node_component_satellite_-orbiting_initialize
subroutine node_component_satellite_-preset_orphanize
module node_component_spheroid_-standard
module node_component_spheroid_very_-simple
subroutine node_component_spin_-vitvitska_initialize_spins
function stellar_luminosities_index_-from_properties
subroutine stellar_luminosities_state_-store
subroutine tensor_r2_d3_sym_dump
subroutine tensor_r2_d3_sym_read_raw
function gelmanrubinconstructorparameters
function adaptivegamma
subroutine annealeddifferentialevolutionupdate
subroutine correlationcorrelationlengthcompute
subroutine historyrestore
subroutine simplerestore

function fixedorbit

subroutine statistics_points_power_-spectrum
subroutine fileread
function fileconstructorinternal

function barkana2001wdmconstructorinternal
subroutine farahifileread

function tinker2008constructorinternal

module spherical_collapse_matter_-dark_energy
subroutine spherical_collape_matter_-lambda_delta_virial_tabulate
subroutine store_table

module virial_density_contrast
subroutine system_command_varstr

subroutine conditionalmassfunctionperform
subroutine evolveforestssuspendtree
program test_diemerkravtsov2014_concentration
program test_nfw96_concentration_dark_energy
program test_zhao2009_flat
program test_zhao2009_open
program test_array_monotonicity
program tests_comoving_distance
program test_cooling_functions
program test_dark_matter_halo_radius_enclosing_mass
program test_dark_matter_profiles_heated
program tests_halo_mass_function_tinker
program test_integration2
program tests_kepler_orbits
program tests_linear_growth_cosmological_constant
program tests_linear_growth_open
program test_correa2015_mah
subroutine testvoidfunc
program tests_power_spectrum
program tests_sigma

program tests_spherical_collapse_dark_energy_omega_zero_point_six
program tests_spherical_collapse_dark_energy_omega_half
program tests_spherical_collapse_dark_energy_lambda
program tests_spherical_collapse_flat
program test_stellar_populations

program test_string_utilities
function intergalacticmediumstateevolveupdate
module io_irate
subroutine iratecopycosmology

function fixedconstructorparameters
subroutine catalogprojectedcorrelationfunctionperform
subroutine evolveforestsresumetree

subroutine halomodelgenerateperform
program tests_io_hdf5

program test_prada2011_concentration

program test_zhao2009_dark_energy
program test_abundances
program tests_bug745815
program test_correa2015_concentration
program tests_cosmic_age
program test_dark_matter_profiles

program test_gaunt_factors

program test_hashes_cryptographic

module test_integration2_functions
program tests_linear_growth_eds
program tests_linear_growth_dark_energy
program test_locks
program test_nodes
program test_parameters
program test_search
program tests_spherical_collapse_dark_energy_eds
program tests_spherical_collapse_dark_energy_omega_zero_point_eight
program tests_spherical_collapse_dark_energy_omega_two_thirds
program tests_spherical_collapse_dark_energy_open
program tests_spherical_collapse_open
program test_stellar_populations_luminosities
program test_vectors
module io_hdf5

function irateconstructor
subroutine iratecopysimulation

subroutine iratereadhalos
subroutine iratewritehalos
module mpi_utilities
subroutine openmp_critical_wait_times
function array_intersection_varying_string
subroutine get_temporary_string
module file_utilities
module hashes
module input_parameters
subroutine allocatearray_character_1d
subroutine allocatearray_complex_c_double_complex_3d
subroutine allocatearray_double_precision_1d
subroutine allocatearray_double_precision_2d
subroutine allocatearray_double_precision_3d
subroutine allocatearray_double_precision_4d
subroutine allocatearray_double_precision_5d
subroutine allocatearray_double_precision_6d
subroutine allocatearray_integer_1d
subroutine allocatearray_integer_2d
subroutine allocatearray_integer_kind_int8_1d
subroutine allocatearray_integer_kind_int8_2d
subroutine allocatearray_logical_1d
subroutine allocatearray_logical_2d
subroutine allocatearray_real_1d
subroutine allocatearray_real_2d
subroutine allocatearray_real_kind_quad_1d
subroutine iratereadsimulation
module io_xml
function mpigetranklabel
module openmp_utilities_data
subroutine get_argument_varying_string
function formatted_date_and_time
function executable_find
function hash_md5
subroutine add_memory_component
subroutine allocatearray_character_1d_kind_int8
subroutine allocatearray_complex_c_double_complex_3d_kind_int8
subroutine allocatearray_double_precision_1d_kind_int8
subroutine allocatearray_double_precision_2d_kind_int8
subroutine allocatearray_double_precision_3d_kind_int8
subroutine allocatearray_double_precision_4d_kind_int8
subroutine allocatearray_double_precision_5d_kind_int8
subroutine allocatearray_double_precision_6d_kind_int8
subroutine allocatearray_integer_1d_kind_int8
subroutine allocatearray_integer_2d_kind_int8
subroutine allocatearray_integer_kind_int8_1d_kind_int8
subroutine allocatearray_integer_kind_int8_2d_kind_int8
subroutine allocatearray_logical_1d_kind_int8
subroutine allocatearray_logical_2d_kind_int8
subroutine allocatearray_real_1d_kind_int8
subroutine allocatearray_real_2d_kind_int8
subroutine allocatearray_real_kind_quad_1d_kind_int8

subroutine deallocatearray_character_1d
subroutine deallocatearray_complex_c_-double_complex_3d
subroutine deallocatearray_double_-precision_1d
subroutine deallocatearray_double_-precision_2d
subroutine deallocatearray_double_-precision_3d
subroutine deallocatearray_double_-precision_4d
subroutine deallocatearray_double_-precision_5d
subroutine deallocatearray_double_-precision_6d
subroutine deallocatearray_integer_1d
subroutine deallocatearray_integer_2d
subroutine deallocatearray_integer_-kind_int8_1d
subroutine deallocatearray_integer_-kind_int8_2d
subroutine deallocatearray_logical_1d
subroutine deallocatearray_logical_2d
subroutine deallocatearray_real_1d
subroutine deallocatearray_real_2d
subroutine deallocatearray_real_kind_-quad_1d
subroutine memory_usage_record
subroutine memory_usage_report
module string_handling
module unit_tests

interface: adjustl
Code lines: 2
Contained by: module iso_varying_string

function: adjustl_
Code lines: 13
Contained by: module iso_varying_string

interface: adjustr
Code lines: 2
Contained by: module iso_varying_string

function: adjustr_
Code lines: 13
Contained by: module iso_varying_string

interface: assignment(=)
Code lines: 3
Contained by: module iso_varying_string

interface: char
Code lines: 3
Contained by: module iso_varying_string

function: char_auto
Code lines: 18
Contained by: module iso_varying_string

function: char_fixed
Code lines: 15
Contained by: module iso_varying_string

subroutine: `destroy_vs`
Description: Destroy a varying string object by deallocating it. Can be necessary to avoid memory leaks in some instances.
Code lines: 6
Contained by: module `iso_varying_string`

interface: `extract`
Code lines: 3
Contained by: module `iso_varying_string`

function: `extract_ch`
Code lines: 30
Contained by: module `iso_varying_string`

function: `extract_vs`
Code lines: 15
Contained by: module `iso_varying_string`

interface: `get`
Code lines: 7
Contained by: module `iso_varying_string`

subroutine: `get_`
Code lines: 51
Contained by: module `iso_varying_string`

subroutine: `get_set_ch`
Code lines: 58
Contained by: module `iso_varying_string`

subroutine: `get_set_vs`
Code lines: 22
Contained by: module `iso_varying_string`

subroutine: `get_unit`
Code lines: 53
Contained by: module `iso_varying_string`

subroutine: `get_unit_set_ch`
Code lines: 60
Contained by: module `iso_varying_string`

subroutine: `get_unit_set_vs`
Code lines: 23
Contained by: module `iso_varying_string`

interface: `iachar`
Code lines: 2
Contained by: module `iso_varying_string`

function: `iachar_`
Code lines: 14
Contained by: module `iso_varying_string`

interface: `ichar`
Code lines: 2
Contained by: module `iso_varying_string`

function: `ichar_`
Code lines: 14
Contained by: module `iso_varying_string`

interface: `index`
Code lines: 4
Contained by: module `iso_varying_string`

function: `index_ch_vs`
Code lines: 16
Contained by: module `iso_varying_string`

function: `index_vs_ch`
Code lines: 16
Contained by: module `iso_varying_string`

function: `index_vs_vs`
Code lines: 16
Contained by: module `iso_varying_string`

interface: `insert`
Code lines: 5
Contained by: module `iso_varying_string`

function: `insert_ch_ch`
Code lines: 20
Contained by: module `iso_varying_string`

function: `insert_ch_vs`
Code lines: 15
Contained by: module `iso_varying_string`

function: `insert_vs_ch`
Code lines: 15
Contained by: module `iso_varying_string`

function: `insert_vs_vs`
Code lines: 15
Contained by: module `iso_varying_string`

interface: `len`

Code lines: 2
Contained by: module iso_varying_string

function: len_
Code lines: 17
Contained by: module iso_varying_string

interface: len_trim
Code lines: 2
Contained by: module iso_varying_string

function: len_trim_
Code lines: 17
Contained by: module iso_varying_string

interface: lge
Code lines: 4
Contained by: module iso_varying_string

function: lge_ch_vs
Code lines: 15
Contained by: module iso_varying_string

function: lge_vs_ch
Code lines: 15
Contained by: module iso_varying_string

function: lge_vs_vs
Code lines: 14
Contained by: module iso_varying_string

interface: lgt
Code lines: 4
Contained by: module iso_varying_string

function: lgt_ch_vs
Code lines: 15
Contained by: module iso_varying_string

function: lgt_vs_ch
Code lines: 15
Contained by: module iso_varying_string

function: lgt_vs_vs
Code lines: 14
Contained by: module iso_varying_string

interface: lle
Code lines: 4

Contained by: module iso_varying_string

function: lle_ch_vs
Code lines: 15
Contained by: module iso_varying_string

function: lle_vs_ch
Code lines: 15
Contained by: module iso_varying_string

function: lle_vs_vs
Code lines: 14
Contained by: module iso_varying_string

interface: llt
Code lines: 4
Contained by: module iso_varying_string

function: llt_ch_vs
Code lines: 15
Contained by: module iso_varying_string

function: llt_vs_ch
Code lines: 15
Contained by: module iso_varying_string

function: llt_vs_vs
Code lines: 14
Contained by: module iso_varying_string

subroutine: load_from_file_vs
Description: Load a varying string object with the contents of a file (specified by fileName).
Code lines: 21
Contained by: module iso_varying_string

subroutine: op_assign_ch_vs
Code lines: 13
Contained by: module iso_varying_string

subroutine: op_assign_vs_ch
Code lines: 18
Contained by: module iso_varying_string

function: op_concat_ch_vs
Code lines: 15
Contained by: module iso_varying_string

function: op_concat_vs_ch
Code lines: 15
Contained by: module iso_varying_string

function: `op_concat_vs_vs`
Code lines: 21
Contained by: module `iso_varying_string`

function: `op_eq_ch_vs`
Code lines: 15
Contained by: module `iso_varying_string`

function: `op_eq_vs_ch`
Code lines: 15
Contained by: module `iso_varying_string`

function: `op_eq_vs_vs`
Code lines: 14
Contained by: module `iso_varying_string`

function: `op_ge_ch_vs`
Code lines: 15
Contained by: module `iso_varying_string`

function: `op_ge_vs_ch`
Code lines: 15
Contained by: module `iso_varying_string`

function: `op_ge_vs_vs`
Code lines: 14
Contained by: module `iso_varying_string`

function: `op_gt_ch_vs`
Code lines: 15
Contained by: module `iso_varying_string`

function: `op_gt_vs_ch`
Code lines: 15
Contained by: module `iso_varying_string`

function: `op_gt_vs_vs`
Code lines: 14
Contained by: module `iso_varying_string`

function: `op_le_ch_vs`
Code lines: 15
Contained by: module `iso_varying_string`

function: `op_le_vs_ch`
Code lines: 15
Contained by: module `iso_varying_string`

function: `op_le_vs_vs`
Code lines: 14
Contained by: module `iso_varying_string`

function: `op_lt_ch_vs`
Code lines: 15
Contained by: module `iso_varying_string`

function: `op_lt_vs_ch`
Code lines: 15
Contained by: module `iso_varying_string`

function: `op_lt_vs_vs`
Code lines: 14
Contained by: module `iso_varying_string`

function: `op_ne_ch_vs`
Code lines: 15
Contained by: module `iso_varying_string`

function: `op_ne_vs_ch`
Code lines: 15
Contained by: module `iso_varying_string`

function: `op_ne_vs_vs`
Code lines: 14
Contained by: module `iso_varying_string`

interface: `operator(`
Code lines: 4
Contained by: module `iso_varying_string`

interface: `operator(//)`
Code lines: 4
Contained by: module `iso_varying_string`

interface: `operator(/=)`
Code lines: 4
Contained by: module `iso_varying_string`

interface: `operator(==)`
Code lines: 4
Contained by: module `iso_varying_string`

interface: `put`
Code lines: 5
Contained by: module `iso_varying_string`

subroutine: `put_ch`
Code lines: 16
Contained by: module `iso_varying_string`

interface: `put_line`
Code lines: 5
Contained by: module `iso_varying_string`

subroutine: `put_line_ch`
Code lines: 18
Contained by: module `iso_varying_string`

subroutine: `put_line_unit_ch`
Code lines: 19
Contained by: module `iso_varying_string`

subroutine: `put_line_unit_vs`
Code lines: 15
Contained by: module `iso_varying_string`

subroutine: `put_line_vs`
Code lines: 14
Contained by: module `iso_varying_string`

subroutine: `put_unit_ch`
Code lines: 19
Contained by: module `iso_varying_string`

subroutine: `put_unit_vs`
Code lines: 15
Contained by: module `iso_varying_string`

subroutine: `put_vs`
Code lines: 12
Contained by: module `iso_varying_string`

interface: `remove`
Code lines: 3
Contained by: module `iso_varying_string`

function: `remove_ch`
Code lines: 34
Contained by: module `iso_varying_string`

function: `remove_vs`
Code lines: 15
Contained by: module `iso_varying_string`

interface: `repeat`
Code lines: 2
Contained by: module `iso_varying_string`

function: `repeat_`
Code lines: 14
Contained by: module `iso_varying_string`

interface: `replace`
Code lines: 17
Contained by: module `iso_varying_string`

function: `replace_ch_ch_auto`
Code lines: 16
Contained by: module `iso_varying_string`

function: `replace_ch_ch_ch_target`
Code lines: 80
Contained by: module `iso_varying_string`

function: `replace_ch_ch_fixed`
Code lines: 27
Contained by: module `iso_varying_string`

function: `replace_ch_ch_vs_target`
Code lines: 19
Contained by: module `iso_varying_string`

function: `replace_ch_vs_auto`
Code lines: 16
Contained by: module `iso_varying_string`

function: `replace_ch_vs_ch_target`
Code lines: 19
Contained by: module `iso_varying_string`

function: `replace_ch_vs_fixed`
Code lines: 17
Contained by: module `iso_varying_string`

function: `replace_ch_vs_vs_target`
Code lines: 19
Contained by: module `iso_varying_string`

function: `replace_vs_ch_auto`
Code lines: 16
Contained by: module `iso_varying_string`

function: `replace_vs_ch_ch_target`
Code lines: 19
Contained by: module `iso_varying_string`

function: `replace_vs_ch_fixed`
Code lines: 17
Contained by: module `iso_varying_string`

function: `replace_vs_ch_vs_target`
Code lines: 19
Contained by: module `iso_varying_string`

function: `replace_vs_vs_auto`
Code lines: 16
Contained by: module `iso_varying_string`

function: `replace_vs_vs_ch_target`
Code lines: 19
Contained by: module `iso_varying_string`

function: `replace_vs_vs_fixed`
Code lines: 17
Contained by: module `iso_varying_string`

function: `replace_vs_vs_vs_target`
Code lines: 19
Contained by: module `iso_varying_string`

interface: `scan`
Code lines: 4
Contained by: module `iso_varying_string`

function: `scan_ch_vs`
Code lines: 23
Contained by: module `iso_varying_string`

function: `scan_vs_ch`
Code lines: 23
Contained by: module `iso_varying_string`

function: `scan_vs_vs`
Code lines: 23
Contained by: module `iso_varying_string`

interface: `split`
Code lines: 3
Contained by: module `iso_varying_string`

subroutine: `split_ch`
Code lines: 49
Contained by: module `iso_varying_string`

subroutine: `split_vs`
Code lines: 21
Contained by: module `iso_varying_string`

interface: `trim`
Code lines: 2
Contained by: module `iso_varying_string`

function: `trim_`
Code lines: 13
Contained by: module `iso_varying_string`

interface: `var_str`
Code lines: 2
Contained by: module `iso_varying_string`

function: `var_str_`
Code lines: 23
Contained by: module `iso_varying_string`

type: `varying_string`
Code lines: 31
Contained by: module `iso_varying_string`

interface: `verify`
Code lines: 4
Contained by: module `iso_varying_string`

function: `verify_ch_vs`
Code lines: 23
Contained by: module `iso_varying_string`

function: `verify_vs_ch`
Code lines: 23
Contained by: module `iso_varying_string`

function: `verify_vs_vs`
Code lines: 23
Contained by: module `iso_varying_string`

subroutine: `vsfinalize`
Description: Destroy a varying string object by deallocating it. Can be necessary to avoid memory leaks in some instances.
Code lines: 6
Contained by: module `iso_varying_string`

subroutine: `vsstaterestore`
Description: Restore the state of a `varying_string` object from file.
Code lines: 17
Contained by: module `iso_varying_string`
Modules used: `iso_c_binding`

subroutine: `vsstatestore`
Description: Store the state of a `varying_string` object to file.
Code lines: 13
Contained by: module `iso_varying_string`
Modules used: `iso_c_binding`

file: `libmatheval_config.cpp`

file: `mass_distributions.F90`
Description: Contains a module which implements a class that provides mass distributions.
Code lines: 96

module: `mass_distributions`
Description: Implements a class that provides mass distributions.
Code lines: 74
Contained by: file `mass_distributions.F90`
Modules used: `coordinates`
Used by:
- subroutine `galacticus_function_classes_destroy`
- subroutine `galacticus_state_retrieve`
- subroutine `galacticus_state_store`
- file `hot_halo.mass_distribution.beta_profile.F90`
- function `node_component_spheroid_standard_half_mass_radius`
- module `node_component_disk_standard_data`
- module `node_component_hot_halo_cold_mode_structure_tasks`
- module `node_component_spheroid_standard_data`
- program `test_mass_distributions`

file: `mass_distributions.cylindrical.F90`
Description: Implementation of an abstract mass distribution class for cylindrically symmetric distributions.
Code lines: 120

function: `cylindricalsymmetry`
Description: Returns symmetry label for mass dsitributions with cylindrical symmetry.
Code lines: 8
Contained by: file `mass_distributions.cylindrical.F90`

type: `massdistributioncylindrical`
Description: Implementation of an abstract mass distribution class for cylindrically symmetric distributions.
Code lines: 43
Contained by: file `mass_distributions.cylindrical.F90`

file: `mass_distributions.cylindrical.Miyamoto_Nagai.F90`
Description: Implementation of an Miyamoto-Nagai model [Miyamoto and Nagai, 1975] mass distribution class.
Code lines: 446
Modules used: `tables`

interface: `massdistributionmiyamotonagai`
Description: Constructors for the `miyamotoNagai` mass distribution class.
Code lines: 4
Contained by: file `mass_distributions.cylindrical.Miyamoto_Nagai.F90`

function: `miyamotonagaiconstructorinternal`
Description: Internal constructor for "miyamotoNagai" mass distribution class.
Code lines: 43
Contained by: file `mass_distributions.cylindrical.Miyamoto_Nagai.F90`
Modules used: `galacticus_error` `numerical_comparison` `numerical_constants_math`

function: `miyamotonagaiconstructorparameters`
Description: Constructor for the `miyamotoNagai` mass distribution class which builds the object from a parameter set.
Code lines: 52
Contained by: file `mass_distributions.cylindrical.Miyamoto_Nagai.F90`
Modules used: `input_parameters` `numerical_constants_math`

function: `miyamotonagaidensity`
Description: Return the density at the specified `coordinates` in an exponential disk mass distribution.
Code lines: 17
Contained by: file `mass_distributions.cylindrical.Miyamoto_Nagai.F90`
Modules used: `coordinates`

function: `miyamotonagaimassenclosedbysphere`
Description: Computes the mass enclosed within a sphere of given `radius` for exponential disk mass distributions.
Code lines: 12
Contained by: file `mass_distributions.cylindrical.Miyamoto_Nagai.F90`
Modules used: `numerical_constants_math`

subroutine: `miyamotonagaimassenclosedtabulate`
Description: Construct a tabulation of the mass enclosed by a sphere in a Miyamoto-Nagai mass distribution.
Code lines: 62
Contained by: file `mass_distributions.cylindrical.Miyamoto_Nagai.F90`
Modules used: `fgsl` `numerical_constants_math` `numerical_integration` `table_labels`

function: `integrandr`
Description: Integrand function used for finding the mass enclosed by a sphere in Miyamoto-Nagai disks.
Code lines: 12

Contained by: subroutine miyamotonagaimassenclosedtabulate
Modules used: coordinates

function: integrandz
Description: Integrand function used for finding the mass enclosed by a sphere in Miyamoto-Nagai disks.
Code lines: 10
Contained by: subroutine miyamotonagaimassenclosedtabulate
Modules used: coordinates

function: miyamotonagaipotential
Description: Return the gravitational potential for an exponential disk.
Code lines: 20
Contained by: file mass_distributions.cylindrical.Miyamoto_Nagai.F90
Modules used: coordinates numerical_constants_physical

function: miyamotonagairadiushalfmass
Description: Return the half-mass radius in a Miyamoto-Nagai mass distribution.
Code lines: 10
Contained by: file mass_distributions.cylindrical.Miyamoto_Nagai.F90

function: miyamotonagairotationcurve
Description: Return the mid-plane rotation curve for a Miyamoto-Nagai mass distribution.
Code lines: 16
Contained by: file mass_distributions.cylindrical.Miyamoto_Nagai.F90
Modules used: coordinates numerical_constants_physical

function: miyamotonagairotationcurvegradient
Description: Return the mid-plane rotation curve gradient for an exponential disk.
Code lines: 16
Contained by: file mass_distributions.cylindrical.Miyamoto_Nagai.F90
Modules used: coordinates numerical_constants_physical

function: miyamotonagaisurfacedensity
Description: Return the surface density at the specified coordinates in a Miyamoto-Nagai mass distribution.
Code lines: 14
Contained by: file mass_distributions.cylindrical.Miyamoto_Nagai.F90
Modules used: coordinates

function: miyamotonagaisurfacedensityradialmoment
Description: Compute radial moments of the Miyamoto-Nagai mass distribution surface density profile.
Code lines: 52
Contained by: file mass_distributions.cylindrical.Miyamoto_Nagai.F90
Modules used: galacticus_error

function: momentweight
Description: The weight function used in computing radial moments of the Miyamoto-Nagai mass distribution surface density.
Code lines: 7
Contained by: function miyamotonagaisurfacedensityradialmoment

subroutine: `miyamotonagaisurfacedensitytabulate`
Description: Construct a tabulation of the surface density profile in a Miyamoto-Nagai mass distribution.
Code lines: 42
Contained by: file `mass_distributions.cylindrical.Miyamoto_Nagai.F90`
Modules used: `fgsl` `numerical_integration` `table_labels`

function: `integrandsurfacedensity`
Description: Integrand function used for finding the surface density of Miyamoto-Nagai disks.
Code lines: 10
Contained by: subroutine `miyamotonagaisurfacedensitytabulate`
Modules used: `coordinates`

file: `mass_distributions.cylindrical.exponential_disk.F90`
Description: Implementation of an exponential disk mass distribution class.
Code lines: 578
Modules used: `tables`

function: `exponentialdiskbesselfactorpotential`
Description: Compute Bessel function factors appearing in the expression for an razor-thin exponential disk gravitational potential.
Code lines: 21
Contained by: file `mass_distributions.cylindrical.exponential_disk.F90`
Modules used: `bessel_functions` `numerical_constants_math`

function: `exponentialdiskbesselfactorrotationcurve`
Description: Compute Bessel function factors appearing in the expression for an razor-thin exponential disk rotation curve.
Code lines: 48
Contained by: file `mass_distributions.cylindrical.exponential_disk.F90`
Modules used: `bessel_functions` `numerical_constants_math`

function: `exponentialdiskbesselfactorrotationcurvegradient`
Description: Compute Bessel function factors appearing in the expression for a razor-thin exponential disk rotation curve gradient.
Code lines: 46
Contained by: file `mass_distributions.cylindrical.exponential_disk.F90`
Modules used: `bessel_functions` `numerical_constants_math`

function: `exponentialdiskconstructorinternal`
Description: Internal constructor for “exponentialDisk” mass distribution class.
Code lines: 61
Contained by: file `mass_distributions.cylindrical.exponential_disk.F90`
Modules used: `galacticus_error` `numerical_comparison` `numerical_constants_math`

function: `exponentialdiskconstructorparameters`
Description: Constructor for the `exponentialDisk` mass distribution class which builds the object from a parameter set.

Code lines: 52
Contained by: file mass_distributions.cylindrical.exponential_disk.F90
Modules used: input_parameters numerical_constants_math

function: exponentialdiskdensity
Description: Return the density at the specified coordinates in an exponential disk mass distribution.
Code lines: 25
Contained by: file mass_distributions.cylindrical.exponential_disk.F90
Modules used: coordinates galacticus_error

subroutine: exponentialdiskdestructor
Description: Destructor for exponential disk mass distributions.
Code lines: 13
Contained by: file mass_distributions.cylindrical.exponential_disk.F90

function: exponentialdiskmassenclosedbysphere
Description: Computes the mass enclosed within a sphere of given radius for exponential disk mass distributions.
Code lines: 11
Contained by: file mass_distributions.cylindrical.exponential_disk.F90
Modules used: numerical_constants_math

function: exponentialdiskpotential
Description: Return the gravitational potential for an exponential disk.
Code lines: 36
Contained by: file mass_distributions.cylindrical.exponential_disk.F90
Modules used: bessel_functions coordinates numerical_constants_physical

function: exponentialdiskradiushalfmass
Description: Return the half-mass radius in an exponential disk mass distribution.
Code lines: 8
Contained by: file mass_distributions.cylindrical.exponential_disk.F90

function: exponentialdiskrotationcurve
Description: Return the mid-plane rotation curve for an exponential disk.
Code lines: 38
Contained by: file mass_distributions.cylindrical.exponential_disk.F90
Modules used: coordinates numerical_constants_physical

function: exponentialdiskrotationcurvegradient
Description: Return the mid-plane rotation curve gradient for an exponential disk.
Code lines: 24
Contained by: file mass_distributions.cylindrical.exponential_disk.F90
Modules used: coordinates numerical_constants_physical

function: exponentialdisksurfacedensity
Description: Return the surface density at the specified coordinates in an exponential disk mass distribution.
Code lines: 13

Contained by: file mass_distributions.cylindrical.exponential_disk.F90
Modules used: coordinates

function: exponentialdisksurfacedensityradialmoment
Description: Compute radial moments of the exponential disk mass distribution surface density profile.
Code lines: 31
Contained by: file mass_distributions.cylindrical.exponential_disk.F90
Modules used: galacticus_error gamma_functions

subroutine: exponentialdisktabulate
Description: Build tables used for exponential disk mass distributions.
Code lines: 25
Contained by: file mass_distributions.cylindrical.exponential_disk.F90
Modules used: bessel_functions table_labels

interface: massdistributionexponentialdisk
Description: Constructors for the exponentialDisk mass distribution class.
Code lines: 4
Contained by: file mass_distributions.cylindrical.exponential_disk.F90

file: mass_distributions.spherical.F90
Description: Implementation of an abstract mass distribution class for spherically symmetric distributions.
Code lines: 120

type: massdistributionspherical
Description: Implementation of an abstract mass distribution class for spherically symmetric distributions.
Code lines: 16
Contained by: file mass_distributions.spherical.F90

function: sphericalmassenclosedbysphere
Description: Computes the mass enclosed within a sphere of given radius for spherically-symmetric mass distributions using numerical integration.
Code lines: 18
Contained by: file mass_distributions.spherical.F90
Modules used: fgsl numerical_constants_math numerical_integration

function: sphericalmassenclosedbysphereintegrand
Description: Enclosed mass integrand for spherical mass distributions.
Code lines: 10
Contained by: file mass_distributions.spherical.F90
Modules used: coordinates

function: sphericalmassroot
Description: Root function used in finding half mass radii of spherically symmetric mass distributions.
Code lines: 7
Contained by: file mass_distributions.spherical.F90

function: `sphericalradiushalfmass`
Description: Computes the half-mass radius of a spherically symmetric mass distribution using numerical root finding.
Code lines: 19
Contained by: file `mass_distributions.spherical.F90`
Modules used: `fgsl` `root_finder`

function: `sphericalsymmetry`
Description: Returns symmetry label for mass dsitributions with spherical symmetry.
Code lines: 8
Contained by: file `mass_distributions.spherical.F90`

file: `mass_distributions.spherical.Hernquist.F90`
Description: Implementation of a Hernquist [1990] mass distribution class.
Code lines: 241

function: `hernquistconstructorinternal`
Description: Internal constructor for "hernquist" mass distribution class.
Code lines: 44
Contained by: file `mass_distributions.spherical.Hernquist.F90`
Modules used: `galacticus_error` `numerical_comparison` `numerical_constants_math`

function: `hernquistconstructorparameters`
Description: Constructor for the `hernquist` mass distribution class which builds the object from a parameter set.
Code lines: 52
Contained by: file `mass_distributions.spherical.Hernquist.F90`
Modules used: `input_parameters` `numerical_constants_math`

function: `hernquistdensity`
Description: Return the density at the specified `coordinates` in a Hernquist mass distribution.
Code lines: 15
Contained by: file `mass_distributions.spherical.Hernquist.F90`
Modules used: `coordinates`

function: `hernquistdensityradialmoment`
Description: Returns a radial density moment for the Hernquist mass distribution.
Code lines: 34
Contained by: file `mass_distributions.spherical.Hernquist.F90`
Modules used: `galacticus_error` `numerical_comparison` `numerical_constants_math`

function: `hernquistmassenclosedbysphere`
Description: Computes the mass enclosed within a sphere of given `radius` for Hernquist mass distributions.
Code lines: 17
Contained by: file `mass_distributions.spherical.Hernquist.F90`
Modules used: `numerical_constants_math`

function: `hernquistpotential`
Description: Return the potential at the specified `coordinates` in a Hernquist mass distribution.
Code lines: 15
Contained by: file `mass_distributions.spherical.Hernquist.F90`
Modules used: `coordinates` `numerical_constants_physical`

function: `hernquistradiushalfmass`
Description: Return the half-mass radius of a Hernquist mass distribution.
Code lines: 8
Contained by: file `mass_distributions.spherical.Hernquist.F90`

interface: `massdistributionhernquist`
Description: Constructors for the `hernquist` mass distribution class.
Code lines: 4
Contained by: file `mass_distributions.spherical.Hernquist.F90`

file: `mass_distributions.spherical.NFW.F90`
Description: Implementation of an NFW [Navarro et al., 1996] mass distribution class.
Code lines: 164

interface: `massdistributionnfw`
Description: Constructors for the `nfw` mass distribution class.
Code lines: 4
Contained by: file `mass_distributions.spherical.NFW.F90`

function: `nfwconstructorinternal`
Description: Internal constructor for "nfw" mass distribution class.
Code lines: 35
Contained by: file `mass_distributions.spherical.NFW.F90`
Modules used: `galacticus_error` `numerical_comparison` `numerical_constants_math`

function: `nfwconstructorparameters`
Description: Constructor for the `nfw` mass distribution class which builds the object from a parameter set.
Code lines: 70
Contained by: file `mass_distributions.spherical.NFW.F90`
Modules used: `input_parameters` `numerical_constants_math`

function: `nfwdensity`
Description: Return the density at the specified `coordinates` in an NFW mass distribution.
Code lines: 15
Contained by: file `mass_distributions.spherical.NFW.F90`
Modules used: `coordinates`

file: `mass_distributions.spherical.Sersic.F90`
Description: Implementation of a Sérsic mass distribution class.
Code lines: 458

Modules used: fgsl

interface: massdistributionsersic
Description: Constructors for the sersic mass distribution class.
Code lines: 4
Contained by: file mass_distributions.spherical.Sersic.F90

function: sersicabelintegrand
Description: The integrand in the Abel integral used to invert the Sérsic profile to get the corresponding 3-D profile.
Code lines: 12
Contained by: file mass_distributions.spherical.Sersic.F90
Modules used: numerical_constants_math

function: sersiccoefficientroot
Description: Root function used in finding the coefficient for Sérsic profiles.
Code lines: 8
Contained by: file mass_distributions.spherical.Sersic.F90
Modules used: gamma_functions

function: sersicconstructorinternal
Description: Internal constructor for "sersic" mass distribution class.
Code lines: 47
Contained by: file mass_distributions.spherical.Sersic.F90
Modules used: galacticus_error numerical_comparison numerical_constants_math

function: sersicconstructorparameters
Description: Constructor for the sersic mass distribution class which builds the object from a parameter set.
Code lines: 51
Contained by: file mass_distributions.spherical.Sersic.F90
Modules used: input_parameters

function: sersicdensity
Description: Return the density at the specified coordinates in a Sérsic mass distribution.
Code lines: 20
Contained by: file mass_distributions.spherical.Sersic.F90
Modules used: coordinates numerical_interpolation table_labels

function: sersicdensityradialmoment
Description: Returns a radial density moment for the Sérsic mass distribution.
Code lines: 40
Contained by: file mass_distributions.spherical.Sersic.F90
Modules used: galacticus_error numerical_comparison numerical_constants_math

function: sersicmassenclosedbysphere
Description: Computes the mass enclosed within a sphere of given radius for Sérsic mass distributions.

Code lines: 23
Contained by: file mass_distributions.spherical.Sersic.F90
Modules used: numerical_constants_math numerical_interpolation

function: sersicpotential
Description: Return the potential at the specified coordinates in a Sérsic mass distribution.
Code lines: 25
Contained by: file mass_distributions.spherical.Sersic.F90
Modules used: coordinates numerical_constants_physical numerical_interpolation

function: sersicradiushalfmass
Description: Return the half-mass radius of a Sérsic mass distribution.
Code lines: 9
Contained by: file mass_distributions.spherical.Sersic.F90

function: sersicradiushalfmassprojected
Description: Return the half-mass radius in projection of a Sérsic mass distribution.
Code lines: 7
Contained by: file mass_distributions.spherical.Sersic.F90

subroutine: sersictabulate
Description: Tabulate the density and enclosed mass in a dimensionless Sérsic profile.
Code lines: 113
Contained by: file mass_distributions.spherical.Sersic.F90
Modules used: memory_management numerical_constants_math numerical_integration numerical_interpolation numerical_ranges root_finder

file: mass_distributions.spherical.beta_profile.F90
Description: Implementation of a β-profile mass distribution class.
Code lines: 404

function: betaprofileconstructorinternal
Description: Constructor for "betaProfile" convergence class.
Code lines: 88
Contained by: file mass_distributions.spherical.beta_profile.F90
Modules used: galacticus_display galacticus_error hypergeometric_functions numerical_comparison numerical_constants_math

function: betaprofileconstructorparameters
Description: Constructor for the betaProfile mass distribution class which builds the object from a parameter set.
Code lines: 63
Contained by: file mass_distributions.spherical.beta_profile.F90
Modules used: input_parameters

function: betaprofiledensity

Description: Return the density at the specified `coordinates` in a β-profile mass distribution.
Code lines: 14
Contained by: file mass_distributions.spherical.beta_profile.F90

function: `betaprofiledensitygradientradial`
Description: Return the density at the specified `coordinates` in a β-profile mass distribution.
Code lines: 23
Contained by: file mass_distributions.spherical.beta_profile.F90

function: `betaprofiledensityradialmoment`
Description: Computes radial moments of the density in a β-profile mass distribution.
Code lines: 98
Contained by: file mass_distributions.spherical.beta_profile.F90
Modules used: hypergeometric_functions numerical_comparison

function: `radialmomenttwothirds`
Description: Special case of radial moment for $\beta = 2/3$ β-profile.
Code lines: 45
Contained by: function betaprofiledensityradialmoment
Modules used: galacticus_error

function: `betaprofilemassenclosedbysphere`
Description: Computes the mass enclosed within a sphere of given `radius` for β-profile mass distributions. Result computed using Wolfram Alpha.
Code lines: 25
Contained by: file mass_distributions.spherical.beta_profile.F90
Modules used: hypergeometric_functions numerical_constants_math

function: `betaprofilepotential`
Description: Return the potential at the specified `coordinates` in a β-profile mass distribution. Calculated using Wolfram Alpha.
Code lines: 36
Contained by: file mass_distributions.spherical.beta_profile.F90
Modules used: coordinates hypergeometric_functions
numerical_comparison numerical_constants_math
numerical_constants_physical

interface: `massdistributionbetaprofile`
Description: Constructors for the `betaProfile` mass distribution class.
Code lines: 4
Contained by: file mass_distributions.spherical.beta_profile.F90

file: `math.Bessel_functions.F90`
Description: Contains a module which implements calculations of Bessel functions.
Code lines: 67

module: `bessel_functions`
Description: Implements calculations of Bessel functions.
Code lines: 45

Contained by: file math.Bessel_functions.F90
Modules used: fgsl
Used by: function exponentialdiskbesselfactorpotential; function exponentialdiskbesselfactorrotationcurve; function exponentialdiskbesselfactorrotationcurvegradient; function exponentialdiskpotential; subroutine exponentialdisktabulate; function node_component_disk_standard_potential; function jiang2014distributionvelocitytangential; program test_math_special_functions

function: bessel_function_i0
Description: Computes the I_0 Bessel function.
Code lines: 7
Contained by: module bessel_functions

function: bessel_function_i1
Description: Computes the I_1 Bessel function.
Code lines: 7
Contained by: module bessel_functions

function: bessel_function_k0
Description: Computes the K_0 Bessel function.
Code lines: 7
Contained by: module bessel_functions

function: bessel_function_k1
Description: Computes the K_1 Bessel function.
Code lines: 7
Contained by: module bessel_functions

file: math.Struve_functions.F90
Description: Contains a module which implements Struve functions.
Code lines: 144

module: struve_functions
Description: Implements Struve functions.
Code lines: 122
Contained by: file math.Struve_functions.F90
Used by: function jiang2014distributionvelocitytangential

function: struve_function_l1
Description: Evaluate and return the Struve L_1 function.
Code lines: 7
Contained by: module struve_functions

subroutine: stvl1
Code lines: 91

Contained by: module struve_functions

file: math.beta_functions.F90
Description: Contains a module which implements beta functions.
Code lines: 68

module: beta_functions
Description: Implements beta functions.
Code lines: 46
Contained by: file math.beta_functions.F90
Modules used: fgsl iso_c_binding
Used by: function betacumulative function betadensity

function: beta_function
Description: Evaluate the beta function, $B(a, b)$.
Code lines: 7
Contained by: module beta_functions

function: beta_function_incomplete_normalized
Description: Evaluate the normalized incomplete beta function, $B_x(a, b)/B(a, b)$.
Code lines: 7
Contained by: module beta_functions

file: math.distributions.Gaussian.F90
Description: Contains a module which implements Gaussian distributions.
Code lines: 40

module: math_distributions_gaussian
Description: Implements Gaussian distributions.
Code lines: 18
Contained by: file math.distributions.Gaussian.F90
Used by: function zhanghuig1 function zhanghuig2
function zhanghuig2integrand

function: gaussian_distribution
Description: Computes the Gaussian distribution with dispersion sigma at argument x.
Code lines: 9
Contained by: module math_distributions_gaussian
Modules used: numerical_constants_math

file: math.distributions.Poisson_binomial.F90
Description: Contains a module which implements Poisson binomial distributions.
Code lines: 140

module: math_distributions_poisson_binomial
Description: Implements Poisson binomial distributions.
Code lines: 118
Contained by: file math.distributions.Poisson_binomial.F90
Used by: subroutine program test_math_distributions
correlationfunctionaccumulatehalo

function: `poisson_binomial_distribution`
Description: Computes the Poisson binomial distribution with event probabilities `p` at argument `k`. Uses the discrete Fourier transform method proposed by Fernandez and Williams [2010].
Code lines: 31
Contained by: module `math_distributions_poisson_binomial`
Modules used: `numerical_constants_math`

function: `poisson_binomial_distribution_jacobian`
Description: Computes the Jacobian of the Poisson binomial distribution with event probabilities `p` at argument `k`. Uses the discrete Fourier transform method proposed by Fernandez and Williams [2010].
Code lines: 31
Contained by: module `math_distributions_poisson_binomial`
Modules used: `numerical_constants_math`

function: `poisson_binomial_distribution_mean`
Description: Computes the mean of a Poisson binomial distribution.
Code lines: 7
Contained by: module `math_distributions_poisson_binomial`

function: `poisson_binomial_distribution_mean_pairs`
Description: Computes the mean number of pairs expected from a Poisson binomial distribution with event probabilities `p`. Assumes that pair order is significant, i.e. both AB and BA are counted.
Code lines: 13
Contained by: module `math_distributions_poisson_binomial`

function: `poisson_binomial_distribution_mean_pairs_jacobian`
Description: Computes the Jacobian of the mean number of pairs expected from a Poisson binomial distribution with event probabilities `p`. Assumes that pair order is significant, i.e. both AB and BA are counted.
Code lines: 10
Contained by: module `math_distributions_poisson_binomial`

function: `poisson_binomial_distribution_variance`
Description: Computes the mean of a Poisson binomial distribution.
Code lines: 7
Contained by: module `math_distributions_poisson_binomial`

file: `math.error_function.F90`
Description: Contains a module which implements calculations of error functions.
Code lines: 130

module: `error_functions`
Description: Implements calculations of error functions.
Code lines: 108
Contained by: file `math.error_function.F90`
Used by: function `normaloperate` function `normaloperate`

function gaussianregressionevaluate
function chandrasekhar1943acceleration
function zentner2005masslossrate
function voightdensity
function normalcumulative
function surfacebrightnesscompleteness
function farahiprobability
function farahimidpointprobability
function normalconstructorinternal
subroutine particleswarmsimulate
function gnedin1999heatingrate
function voightcumulative
function normalconstructorinternal
function peakbackgroundconstructorinternal
function chabrier2001constructorinternal
subroutine farahiratetabulate
subroutine farahimidpointratetabulate
program test_math_special_functions

interface: `erfapproximate`
Code lines: 2
Contained by: module error_functions

function: `erfapproximatequad`
Description: An approximation to the error function that is designed to be very accurate in the vicinity of zero and infinity.
Code lines: 18
Contained by: module error_functions
Modules used: kind_numbers numerical_constants_math

interface: `error_function`
Code lines: 3
Contained by: module error_functions

interface: `error_function_complementary`
Code lines: 3
Contained by: module error_functions

function: `error_function_complementary_complex`
Description: Computes the complex complementary error function, using the algorithm of Abrarov and Quine [2013].
Code lines: 26
Contained by: module error_functions
Modules used: numerical_constants_math

function: `error_function_complementary_real`
Description: Computes the complementary error function.
Code lines: 8
Contained by: module error_functions
Modules used: fgsl

function: `error_function_complex`
Description: Computes the complex complementary error function.
Code lines: 7
Contained by: module error_functions

function: `error_function_real`
Description: Computes the error function.
Code lines: 8
Contained by: module `error_functions`
Modules used: `fgsl`

function: `faddeeva`
Description: The Fadeeva function.
Code lines: 7
Contained by: module `error_functions`

file: `math.exponential_integrals.F90`
Description: Contains a module which implements exponential integrals.
Code lines: 193

module: `exponential_integrals`
Description: Implements exponential integrals.
Code lines: 171
Contained by: file `math.exponential_integrals.F90`
Modules used: `fgsl`
Used by: function `burkertkspace`, function `isothermalkspace`, function `nfwkspace`, program `test_math_special_functions`

function: `cosine_integral`
Description: Evaluate the $\mathrm{Ci}(x) \equiv \int_0^x \mathrm{d}t \cos(t)/t$ cosine integral.
Code lines: 7
Contained by: module `exponential_integrals`

subroutine: `e1z`
Code lines: 85
Contained by: module `exponential_integrals`

interface: `exponential_integral`
Code lines: 3
Contained by: module `exponential_integrals`

function: `exponential_integral_double`
Description: Exponential integral for real argument `x`.
Code lines: 7
Contained by: module `exponential_integrals`

function: `exponential_integral_double_complex`
Description: Exponential integral, $E_\mathrm{i}(z)$, for complex argument `z`.
Code lines: 9
Contained by: module `exponential_integrals`
Modules used: `numerical_constants_math`

function: `sine_integral`

Description: Evaluate the $\mathrm{Si}(x) \equiv \int_0^x \mathrm{d}t \sin(t)/t$ sine integral.
Code lines: 7
Contained by: module exponential_integrals

file: math.exponentiation.F90
Description: Contains a module which provides fast exponentiation utilizing tables.
Code lines: 113

module: math_exponentiation
Description: Provides a fast exponentiation class which utilizes tables to do rapid exponentiation in a limited range of argument for fixed exponent, along with other exponentiation functions.
Code lines: 91
Contained by: file math.exponentiation.F90
Modules used: tables
Used by:
function naozbarkana2007filteredfractionrate
function virialdensitycontrastdefinitionvirialradiusgu
function diemerkravtsov2014concentrationmean
file hot_halo.outflow_-reincorporation.velocity_maximum_-scaling.F90
file star_-formation.feedback.disks.velocity_-maximum_scaling.F90
file star_formation.rate_surface_-density.disks.Krumholz2009.F90
file star_-formation.timescales.spheroids.velocity_-maximum_scaling.F90
file cooling.cooling_rate.velocity_-maximum_scaling.F90
file dark_matter_profiles.adiabatic_-Gnedin2004.F90
function dark_matter_profile_mass_-definition
module node_component_disk_very_simple
file star_-formation.feedback.spheroids.velocity_-maximum_scaling.F90
file star_-formation.timescales.disks.velocity_-maximum_scaling.F90
program test_math_fast

function: cuberoot
Description: Utilize the fast cbrt() function from the standard C library for computing cube roots.
Code lines: 16
Contained by: module math_exponentiation
Modules used: iso_c_binding

interface: fastexponentiator
Code lines: 2
Contained by: module math_exponentiation

function: fastexponentiatorconstructor
Description: Constructor for the fast exponentiator class.
Code lines: 14
Contained by: module math_exponentiation

function: fastexponentiatorexponentiate

Description: Evaluate the result of an exponentiation operation.
Code lines: 20
Contained by: module math_exponentiation
Modules used: galacticus_error

file: math.factorial.F90
Description: Contains a module which implements calculations of factorials.
Code lines: 55

module: factorials
Description: Implements calculations of factorials
Code lines: 33
Contained by: file math.factorial.F90
Used by: subroutine informationcontentoperate; function polynomialcoefficientcount; function binomialmass; function inoue2014multiplier; function meiksin2006multiplier; program test_math_special_functions

function: factorial
Description: Computes the factorial of argument.
Code lines: 8
Contained by: module factorials
Modules used: fgsl

function: logarithmic_double_factorial
Description: Computes the natural logarithm of the double factorial, $k!!$.
Code lines: 13
Contained by: module factorials

file: math.gamma_function.F90
Description: Contains a module which implements calculations of gamma functions.
Code lines: 113

module: gamma_functions
Description: Implements calculations of gamma functions.
Code lines: 91
Contained by: file math.gamma_function.F90
Used by: function bett2007constructorinternal; function einastocircularvelocitypeakradius; function einastodensityscalefree; function einastoenclosedmassscalefree; function einastofreefallradius; function einastofreefallradiusincreaserate; function einastopotentialscalefree; function einastoradialmoment; function einastoradialvelocitydispersion; function einastojeansequationintegrand; subroutine einastoradiusfromspecificangularmomentumtablemake; function einastorotationnormalization; function exponentialdisksurfacedensityradialmoment; function sersiccoefficientroot

function gammacumulative
function gammadensity
function gammainverse
function negativebinomialmass
function negativebinomialmasslogarithmic
function inoue2014multiplier
function meiksin2006multiplier
function giocoli2008integrated
function massfunctioncovarianceangularpowerradialterm
program test_math_special_functions

function: gamma_function
Description: Computes the gamma function.
Code lines: 7
Contained by: module gamma_functions

function: gamma_function_incomplete
Description: Computes the incomplete gamma function.
Code lines: 8
Contained by: module gamma_functions
Modules used: fgsl

function: gamma_function_incomplete_complementary
Description: Computes the complementary incomplete gamma function.
Code lines: 8
Contained by: module gamma_functions
Modules used: fgsl

function: gamma_function_logarithmic
Description: Computes the logarithm of the gamma function.
Code lines: 8
Contained by: module gamma_functions
Modules used: fgsl

function: inverse_gamma_function_incomplete
Description: Returns the inverse of the incomplete function. That is, it returns x given $Q(a, x)$.
Code lines: 9
Contained by: module gamma_functions

function: inverse_gamma_function_incomplete_complementary
Description: Returns the inverse of the incomplete function. That is, it returns x given $P(a, x)$.
Code lines: 31
Contained by: module gamma_functions
Modules used: galacticus_error incomplete_gamma iso_varying_string

file: math.hypergeometric_functions.F90
Description: Contains a module which implements hypergeometric functions.
Code lines: 146

module: hypergeometric_functions
Description: Implements hypergeometric functions.

Code lines:	121	
Contained by:	file math.hypergeometric_functions.F90	
Modules used:	fgsl	iso_c_binding
Used by:	function radialmoment	function nfwradialmomentscalefree
	function betaprofileconstructorinternal	function betaprofiledensityradialmoment
	function betaprofilemassenclosedbysphere	function betaprofilepotential
	function parkinsoncolehellyfractionsubresolution	function parkinsoncolehellyprobabilitybound
	subroutine parkinsoncolehellysubresolutionhypergeometrictabula	subroutine parkinsoncolehellyupperboundhypergeometrictabula
	function wetzel2010circularitycumulativeprobability	function wetzel2010constructorinternal
	function voightcumulative	function massfunctioncovarianceangularpowerradialterm
	program test_math_special_functions	

function: hypergeometric_1f1

Description: Evaluate the ${}_1F_1(a_1; b_1; x)$ hypergeometric function.
Code lines: 7
Contained by: module hypergeometric_functions

function: hypergeometric_2f1

Description: Evaluate the ${}_2F_1(a_1, a_2; b_1; x)$ hypergeometric function.
Code lines: 51
Contained by: module hypergeometric_functions
Modules used: galacticus_error

interface: hypergeometric_pfq

Code lines: 3
Contained by: module hypergeometric_functions

function: hypergeometric_pfq_complex

Description: Evaluate the generalized hypergeometric function ${}_pF_q(a_1, \ldots, a_p; b_1, \ldots, b_q; x)$, using the algorithm of Perger et al. [1993].
Code lines: 18
Contained by: module hypergeometric_functions

function: hypergeometric_pfq_real

Description: Evaluate the generalized hypergeometric function ${}_pF_q(a_1, \ldots, a_p; b_1, \ldots, b_q; x)$ for real arguments.
Code lines: 8
Contained by: module hypergeometric_functions

file: math.linear_algebra.F90

Description: Contains a module which implements linear algebra calculations.
Code lines: 514

module: linear_algebra

Description: Implements linear algebra calculations.
Code lines: 492
Contained by: file math.linear_algebra.F90
Modules used: fgsl
Used by: subroutine correlationfunctionaccumulatehalo
subroutine correlationfunctionfinalizeanalysis
function correlationfunctionloglikelihood
function meanfunction1dloglikelihood
function scatterfunction1dloglikelihood
function volumefunction1dloglikelihood
subroutine perturbmassesoperate
file models.likelihoods.multivariate_normal.F90
file models.likelihoods.projected_correlation_function.F90

subroutine: arraytomatrixassign
Description: Assign an array to a matrix.
Code lines: 9
Contained by: module linear_algebra

subroutine: arraytovectorassign
Description: Assign an array to a vector.
Code lines: 9
Contained by: module linear_algebra

interface: assignment(=)
Code lines: 6
Contained by: module linear_algebra

type: matrix
Description: Matrix class.
Code lines: 79
Contained by: module linear_algebra

subroutine: matrixcholeskydecompose
Description: Find the Cholesky decomposition of a matrix.
Code lines: 14
Contained by: module linear_algebra

function: matrixcovarianceproduct
Description: Compute the quantity $yC^{-1}y^{\mathrm{T}}$ as appears in likelihood functions utilizing covariance matrices. Instead of directly inverting the covariance matrix (which is computationally slow and can be inaccurate), we solve the linear system $y = Cx$ for x, and then evaluate yx.
Code lines: 10
Contained by: module linear_algebra

subroutine: matrixdestroy
Description: Destroy a matrix object.
Code lines: 7
Contained by: module linear_algebra

function: `matrixdeterminant`
Description: Return the of a matrix.
Code lines: 22
Contained by: module `linear_algebra`

subroutine: `matrixeigensystem`
Description: Find eigenvectors and eigenvalues of a real symmetric matrix.
Code lines: 34
Contained by: module `linear_algebra`

function: `matrixinvert`
Description: Invert a matrix.
Code lines: 29
Contained by: module `linear_algebra`

function: `matrixlinearsystemsolve`
Description: Solve the linear system $y = A \cdot x$.
Code lines: 32
Contained by: module `linear_algebra`

function: `matrixlogarithmicdeterminant`
Description: Return the logarithm of the determinant of a matrix.
Code lines: 22
Contained by: module `linear_algebra`

subroutine: `matrixmakesemipositivedefinite`
Description: Make a matrix semi-positive definite by setting any negative eigenvalues to zero and reconstructing the matrix from its eigenvectors.
Code lines: 29
Contained by: module `linear_algebra`

function: `matrixmatrixmultiply`
Description: Multiply a matrix by a matrix, returning a matrix.
Code lines: 11
Contained by: module `linear_algebra`
Modules used: `galacticus_error`

subroutine: `matrixsymmetrize`
Description: Symmetrize a matrix.
Code lines: 7
Contained by: module `linear_algebra`

subroutine: `matrixtoarrayassign`
Description: Assign a matrix to an array.
Code lines: 8
Contained by: module `linear_algebra`

function: `matrixtranspose`

Description: Transpose a matrix.
Code lines: 9
Contained by: module `linear_algebra`

function: `matrixvectormultiply`
Description: Multiply a matrix by a vector, returning a vector.
Code lines: 13
Contained by: module `linear_algebra`

interface: `operator(*)`
Code lines: 4
Contained by: module `linear_algebra`

type: `vector`
Description: Vector class.
Code lines: 25
Contained by: module `linear_algebra`

function: `vectoradd`
Description: Add one vector to another.
Code lines: 9
Contained by: module `linear_algebra`

subroutine: `vectordestroy`
Description: Destroy a vector object.
Code lines: 7
Contained by: module `linear_algebra`

function: `vectorsubtract`
Description: Subtract one vector from another.
Code lines: 9
Contained by: module `linear_algebra`

subroutine: `vectortoarrayassign`
Description: Assign a vector to an array.
Code lines: 8
Contained by: module `linear_algebra`

subroutine: `vectortovectorassign`
Description: Assign a vector to an array.
Code lines: 9
Contained by: module `linear_algebra`

function: `vectorvectormultiply`
Description: Multiply a vector by a vector, returning a scalar.
Code lines: 8
Contained by: module `linear_algebra`

file: `math.operators.unary.F90`

Description: Contains a module that implements a class of parameter mapping functions.
Code lines: 45

module: `math_operators_unary`
Description: Implements a class of unary operators.
Code lines: 23
Contained by: file `math.operators.unary.F90`
Used by: subroutine `galacticus_function_classes_destroy`; subroutine `galacticus_state_retrieve`; subroutine `galacticus_state_store`; file `models.parameters.active.F90`

file: `math.operators.unary.identity.F90`
Description: Implementation of an identity unary operator.
Code lines: 73

function: `identityconstructorparameters`
Description: Constructor for the `identity` 1D distribution function class which builds the object from a parameter set.
Code lines: 11
Contained by: file `math.operators.unary.identity.F90`
Modules used: `input_parameters`

function: `identityoperate`
Description: Apply an identity operation.
Code lines: 9
Contained by: file `math.operators.unary.identity.F90`

function: `identityunoperate`
Description: Unapply an identity operation.
Code lines: 9
Contained by: file `math.operators.unary.identity.F90`

interface: `operatorunaryidentity`
Description: Constructors for the `identity` 1D distribution function class.
Code lines: 3
Contained by: file `math.operators.unary.identity.F90`

file: `math.operators.unary.inverse.F90`
Description: Implementation of an inverse unary operator.
Code lines: 73

function: `inverseconstructorparameters`
Description: Constructor for the `inverse` 1D distribution function class which builds the object from a parameter set.
Code lines: 11
Contained by: file `math.operators.unary.inverse.F90`
Modules used: `input_parameters`

function: `inverseoperate`

Description: Apply an inverse operation.
Code lines: 9
Contained by: file math.operators.unary.inverse.F90

function: inverseunoperate
Description: Unapply an inverse operation.
Code lines: 9
Contained by: file math.operators.unary.inverse.F90

interface: operatorunaryinverse
Description: Constructors for the inverse 1D distribution function class.
Code lines: 3
Contained by: file math.operators.unary.inverse.F90

file: math.operators.unary.logarithm.F90
Description: Implementation of a logarithm unary operator.
Code lines: 80

function: logarithmconstructorparameters
Description: Constructor for the logarithm 1D distribution function class which builds the object from a parameter set.
Code lines: 11
Contained by: file math.operators.unary.logarithm.F90
Modules used: input_parameters

function: logarithmoperate
Description: Apply an logarithm operation.
Code lines: 9
Contained by: file math.operators.unary.logarithm.F90

function: logarithmunoperate
Description: Unapply an logarithm operation.
Code lines: 13
Contained by: file math.operators.unary.logarithm.F90

interface: operatorunarylogarithm
Description: Constructors for the logarithm 1D distribution function class.
Code lines: 3
Contained by: file math.operators.unary.logarithm.F90

file: math.trigonometric_functions.F90
Description: Contains a module which implements trigonometric functions.
Code lines: 76

module: trigonometric_functions
Description: Implements trigonometric functions.
Code lines: 54
Contained by: file math.trigonometric_functions.F90
Used by: function squareconstructorinternal

interface: `cosec`
Code lines: 3
Contained by: module `trigonometric_functions`

function: `cosecdouble`
Description: Implements cosecant for double precision `x`.
Code lines: 7
Contained by: module `trigonometric_functions`

function: `cosecdoublecomplex`
Description: Implements cosecant for double precision complex `x`.
Code lines: 7
Contained by: module `trigonometric_functions`

interface: `cot`
Code lines: 3
Contained by: module `trigonometric_functions`

function: `cotdouble`
Description: Implements cotangent for double precision `x`.
Code lines: 7
Contained by: module `trigonometric_functions`

function: `cotdoublecomplex`
Description: Implements cotangent for double precision complex `x`.
Code lines: 7
Contained by: module `trigonometric_functions`

file: `math.vector.F90`
Description: Contains a module which implements calculations of vectors.
Code lines: 157

module: `vectors`
Description: Implements calculations of vectors.
Code lines: 135
Contained by: file `math.vector.F90`
Used by:
subroutine `correlationfunctionaccumulatehalo`
subroutine `correlationfunctionfinalizeanalysis`
function `squareconstructorinternal`
function `squareisinlightcone`
subroutine `squarereplicants`
function `cappointincluded`
function `polygonpointincluded`
function `bernardi2013sdsspointincluded`
subroutine `martin2010alfalfarandomsinitialize`
function `fullskypointincluded`
subroutine `fullskywindowfunctions`
function `manglepointincluded`
subroutine `randompointswindowfunctions`
subroutine `triaxialitymodify`
function `readconstruct`
function `readorbitconstruct`
subroutine `readscanformergers`
subroutine `readtimeuntilmergingsubresolution`

subroutine galacticusimport
subroutine points_rotate
subroutine node_component_satellite_orbiting_rate_compute
function chandrasekhar1943acceleration
function gnedin1999heatingrate
subroutine statistics_points_correlation
program test_vectors
function lightconeextract
function kepler_orbits_velocity
module node_component_spin_vitvitska
function spincorrelatedorbit
function zentner2005masslossrate
subroutine pointstoredshiftspace

function: matrix_copy_upper_to_lower_triangle
Description: Copies the upper triangle of a square matrix to the lower triangle.
Code lines: 16
Contained by: module vectors
Modules used: galacticus_error

function: vector_magnitude
Description: Computes the magnitude of vector1.
Code lines: 7
Contained by: module vectors

function: vector_matrix_multiply
Description: Returns the product of a vector with a matrix.
Code lines: 10
Contained by: module vectors

interface: vector_outer_product
Code lines: 3
Contained by: module vectors

interface: vector_outer_product_accumulate
Code lines: 2
Contained by: module vectors

subroutine: vector_outer_product_accumulate_self
Description: Compute the outer product of a vector with itself and accumulate it to the given matrix. Compute only the upper triangle unless the symmetrize option is set to true. If the sparse option is set to true, assume a sparse matrix and accumulate only non-zero terms.
Code lines: 39
Contained by: module vectors

function: vector_outer_product_distinct
Description: Returns the outer product of two vectors.
Code lines: 10
Contained by: module vectors

function: vector_outer_product_self
Description: Returns the outer product of a vector with itself.
Code lines: 12

Contained by: module vectors

function: vector_product
Description: Computes the vector product of vector1 and vector2.
Code lines: 10
Contained by: module vectors

file: merger_trees.branching_probability.F90
Description: Contains a module which implements a merger tree branching probability class.
Code lines: 82

module: merger_tree_branching
Description: Implements a merger tree branching probability class.
Code lines: 60
Contained by: file merger_trees.branching_probability.F90
Modules used: galacticus_nodes, pseudo_random
Used by: subroutine galacticus_function_classes_destroy, subroutine galacticus_state_retrieve, subroutine galacticus_state_store, file merger_trees.construct.builder.Cole2000.F90

file: merger_trees.branching_probability.Parkinson_Cole_Helly.F90
Description: Implements a merger tree branching probability class using the algorithm of Parkinson et al. [2008].
Code lines: 750
Modules used: cosmological_density_field, tables

interface: mergertreebranchingprobabilityparkinsoncolehelly
Description: Constructors for the parkinsonColeHelly merger tree builder class.
Code lines: 4
Contained by: file merger_trees.branching_probability.Parkinson_Cole_Helly.F90

subroutine: parkinsoncolehellycomputecommonfactors
Description: Precomputes some useful factors that are used in the modified Press-Schechter branching integrals.
Code lines: 15
Contained by: file merger_trees.branching_probability.Parkinson_Cole_Helly.F90
Modules used: numerical_constants_math

function: parkinsoncolehellyconstructorinternal
Description: Internal constructor for the "parkinsonColeHelly" merger tree branching probability class.
Code lines: 18
Contained by: file merger_trees.branching_probability.Parkinson_Cole_Helly.F90
Modules used: galacticus_error

function: parkinsoncolehellyconstructorparameters
Description: Constructor for the "parkinsonColeHelly" merger tree branching probability class which reads parameters from a provided parameter list.
Code lines: 73
Contained by: file merger_trees.branching_probability.Parkinson_Cole_Helly.F90

subroutine: `parkinsoncolehellydestructor`
Code lines: 8
Contained by: file `merger_trees.branching_probability.Parkinson_Cole_Helly.F90`

function: `parkinsoncolehellyfractionsubresolution`
Description: Return the fraction of mass accreted in subresolution halos, i.e. those below `massResolution`, per unit change in $\delta_{\rm crit}$ for a halo of mass `haloMass` at time `deltaCritical`. The integral is computed analytically in terms of the ${}_2F_1$ hypergeometric function.
Code lines: 37
Contained by: file `merger_trees.branching_probability.Parkinson_Cole_Helly.F90`
Modules used: `hypergeometric_functions` `numerical_constants_math`

function: `parkinsoncolehellymassbranch`
Description: A merger tree branch split mass function.
Code lines: 127
Contained by: file `merger_trees.branching_probability.Parkinson_Cole_Helly.F90`

function: `massbranchcdmassumptions`
Description: A merger tree branch split mass function which assumes a CDM-like power spectrum. With these assumptions, it can employ the mass sampling algorithm of Parkinson et al. [2008]. One difference with respect to the algorithm of Parkinson et al. [2008] is that here the normalization of their function $S(q)$ (eqn. A2) is irrelevant, since a branch split has already been decided to have occcurred—all that remains necessary is to determine its mass. Variable and function names follow Parkinson et al. [2008].
Code lines: 39
Contained by: function `parkinsoncolehellymassbranch`
Modules used: `pseudo_random`

function: `massbranchgeneric`
Description: Determine the mass of one of the halos to which the given halo branches, given the branching probability, `probability`. Typically, `probabilityFraction` is found by multiplying `probability` by a random variable drawn in the interval 0–1 if a halo branches. This routine then finds the progenitor mass corresponding to this value.
Code lines: 42
Contained by: function `parkinsoncolehellymassbranch`
Modules used: `fgsl` `pseudo_random` `root_finder`

function: `r`
Description: The function $R(q)$ from [Parkinson et al., 2008, eqn. A3].
Code lines: 9
Contained by: function `parkinsoncolehellymassbranch`

function: `v`
Description: The function $V(q)$ from [Parkinson et al., 2008, eqn. A4].
Code lines: 9
Contained by: function `parkinsoncolehellymassbranch`

function: `parkinsoncolehellymassbranchroot`

Description: Used to find the mass of a merger tree branching event.
Code lines: 21
Contained by: file `merger_trees.branching_probability.Parkinson_Cole_Helly.F90`
Modules used: `fgsl` `numerical_integration`

function: `parkinsoncolehellymassbranchrootderivative`

Description: Used to find the mass of a merger tree branching event.
Code lines: 11
Contained by: file `merger_trees.branching_probability.Parkinson_Cole_Helly.F90`
Modules used: `galacticus_error` `numerical_integration`

function: `parkinsoncolehellymergingrate`

Description: Merging rate from Press-Schechter. The constant factor of sqrt(2/pi) not included here—instead it is included in a multiplicative prefactor by which integrals over this function are multiplied.
Code lines: 14
Contained by: file `merger_trees.branching_probability.Parkinson_Cole_Helly.F90`

function: `parkinsoncolehellymodifier`

Description: Empirical modification of the progenitor mass function from Parkinson et al. [2008]. The constant factors of $G_0(\delta_\mathrm{p}/\sigma_\mathrm{p})^{\gamma_2}$ and $1/\sigma_\mathrm{p}^{\gamma_1}$ are not included here—instead they are included in a multiplicative prefactor by which integrals over this function are multiplied.
Code lines: 9
Contained by: file `merger_trees.branching_probability.Parkinson_Cole_Helly.F90`

function: `parkinsoncolehellyprobability`

Description: Return the probability per unit change in δ_crit that a halo of mass `haloMass` at time `deltaCritical` will undergo a branching to progenitors with mass greater than `massResolution`.
Code lines: 33
Contained by: file `merger_trees.branching_probability.Parkinson_Cole_Helly.F90`
Modules used: `fgsl` `numerical_integration`

function: `parkinsoncolehellyprobabilitybound`

Description: Return a bound on the probability per unit change in δ_crit that a halo of mass `haloMass` at time `deltaCritical` will undergo a branching to progenitors with mass greater than `massResolution`.
Code lines: 140
Contained by: file `merger_trees.branching_probability.Parkinson_Cole_Helly.F90`
Modules used: `fgsl` `galacticus_display` `galacticus_error` `hypergeometric_functions` `numerical_comparison` `numerical_constants_math`

function: `parkinsoncolehellyprobabilityintegrandlogarithmic`

Description: Integrand for the branching probability.
Code lines: 10
Contained by: file `merger_trees.branching_probability.Parkinson_Cole_Helly.F90`

function: `parkinsoncolehellyprogenitormassfunction`
Description: Progenitor mass function from Press-Schechter. The constant factor of the parent halo mass is not included here—instead it is included in a multiplicative prefactor by which integrals over this function are multiplied.
Code lines: 8
Contained by: file `merger_trees.branching_probability.Parkinson_Cole_Helly.F90`

function: `parkinsoncolehellystepmaximum`
Description: Return the maximum allowed step in $\delta_{\rm crit}$ that a halo of mass `haloMass` at time `deltaCritical` should be allowed to take.
Code lines: 19
Contained by: file `merger_trees.branching_probability.Parkinson_Cole_Helly.F90`

subroutine: `parkinsoncolehellysubresolutionhypergeometrictabulate`
Description: Tabulate the hypergeometric term appearing in the subresolution merger fraction expression.
Code lines: 46
Contained by: file `merger_trees.branching_probability.Parkinson_Cole_Helly.F90`
Modules used: `galacticus_error` `hypergeometric_functions` `numerical_constants_math` `table_labels`

subroutine: `parkinsoncolehellyupperboundhypergeometrictabulate`
Description: Tabulate the hypergeometric term appearing in the upper bound branching probability rate expression.
Code lines: 54
Contained by: file `merger_trees.branching_probability.Parkinson_Cole_Helly.F90`
Modules used: `galacticus_error` `hypergeometric_functions` `numerical_constants_math` `table_labels`

file: `merger_trees.branching_probability.generalized_Press_Schechter.F90`
Description: Implements a merger tree branching probability class using a generalized Press-Schechter approach.
Code lines: 469
Modules used: `cosmological_density_field` `cosmology_functions` `excursion_sets_first_crossings` `merger_tree_branching_modifiers`

subroutine: `generalizedpressschechtercomputecommonfactors`
Description: Precomputes some useful factors that are used in the generalized Press-Schechter branching integrals.
Code lines: 16
Contained by: file `merger_trees.branching_probability.generalized_Press_Schechter.F90`
Modules used: `cosmological_density_field`

function: `generalizedpressschechterconstructorinternal`
Description: Internal constructor for the Cole et al. [2000] merger tree building class.
Code lines: 18
Contained by: file `merger_trees.branching_probability.generalized_Press_Schechter.F90`
Modules used: `galacticus_error`

function: `generalizedpressschechterconstructorparameters`
Description: Constructor for the "generalizedPressSchechter" merger tree branching probability class which reads parameters from a provided parameter list.
Code lines: 51
Contained by: file `merger_trees.branching_probability.generalized_Press_Schechter.F90`

subroutine: `generalizedpressschechterdestructor`
Code lines: 10
Contained by: file `merger_trees.branching_probability.generalized_Press_Schechter.F90`

subroutine: `generalizedpressschechterexcursionsettest`
Description: Make a call to excursion set routines with the maximum σ that we will use to ensure that they can handle it.
Code lines: 18
Contained by: file `merger_trees.branching_probability.generalized_Press_Schechter.F90`

function: `generalizedpressschechterfractionsubresolution`
Description: Return the fraction of mass accreted in subresolution halos, i.e. those below `massResolution`, per unit change in $\delta_{\rm crit}$ for a halo of mass `haloMass` at time `deltaCritical`. The integral is computed numerically.
Code lines: 57
Contained by: file `merger_trees.branching_probability.generalized_Press_Schechter.F90`
Modules used: `fgsl` `galacticus_display` `galacticus_error` `iso_varying_string` `numerical_integration`

function: `generalizedpressschechterfractionsubresolutionintegrand`
Description: Integrand for the subresolution fraction.
Code lines: 13
Contained by: file `merger_trees.branching_probability.generalized_Press_Schechter.F90`

function: `generalizedpressschechtermassbranch`
Description: Determine the mass of one of the halos to which the given halo branches, given the branching probability, `probabilityFraction`. Typically, `probabilityFraction` is found by multiplying `Generalized_Press_Schechter_Branching_Probability()` by a random variable drawn in the interval 0–1 if a halo branches. This routine then finds the progenitor mass corresponding to this value.
Code lines: 63
Contained by: file `merger_trees.branching_probability.generalized_Press_Schechter.F90`
Modules used: `galacticus_display` `galacticus_error` `iso_varying_string` `pseudo_random` `root_finder`

function: `generalizedpressschechtermassbranchroot`
Description: Root function used in solving for the branch mass.
Code lines: 12
Contained by: file `merger_trees.branching_probability.generalized_Press_Schechter.F90`
Modules used: `fgsl` `numerical_integration`

function: `generalizedpressschechtermergingrate`

Description: Computes the merging rate of dark matter halos in the generalized Press-Schechter algorithm. This "merging rate" is specifically defined as

$$\frac{\mathrm{d}^2 f}{\mathrm{d}\ln M_{\rm child}\mathrm{d}\delta_{\rm c}} = 2\sigma^2(M_{\rm child})\left.\frac{\mathrm{d}\ln\sigma}{\mathrm{d}\ln M}\right|_{M=M_{\rm child}}\frac{\mathrm{d}t}{\mathrm{d}\delta_{\rm c}}\frac{\mathrm{d}f_{12}}{\mathrm{d}t}, \tag{18.21}$$

where $\mathrm{d}f_{12}/\mathrm{d}t$ is the excursion set barrier crossing probabilty per unit time for the effective barrier $B'(S_{\rm child}|S_{\rm parent},t) \equiv B(S_{\rm child},t-\delta t) - B(S_{\rm parent},t)$ in the limit $\delta t \rightarrow 0$.

Code lines: 16

Contained by: file `merger_trees.branching_probability.generalized_Press_Schechter.F90`

function: `generalizedpressschechterprobability`

Description: Return the probability per unit change in $\delta_{\rm crit}$ that a halo of mass `haloMass` at time `deltaCritical` will undergo a branching to progenitors with mass greater than `massResolution`.

Code lines: 27

Contained by: file `merger_trees.branching_probability.generalized_Press_Schechter.F90`

Modules used: `fgsl` `numerical_integration`

function: `generalizedpressschechterprobabilitybound`

Description: Return bounds onthe probability per unit change in $\delta_{\rm crit}$ that a halo of mass `haloMass` at time `deltaCritical` will undergo a branching to progenitors with mass greater than `massResolution`.

Code lines: 13

Contained by: file `merger_trees.branching_probability.generalized_Press_Schechter.F90`

function: `generalizedpressschechterprobabilityintegrand`

Description: Integrand for the branching probability.

Code lines: 9

Contained by: file `merger_trees.branching_probability.generalized_Press_Schechter.F90`

function: `generalizedpressschechterprogenitormassfunction`

Description: Progenitor mass function from Press-Schechter.

Code lines: 8

Contained by: file `merger_trees.branching_probability.generalized_Press_Schechter.F90`

function: `generalizedpressschechterstepmaximum`

Description: Return the maximum allowed step in $\delta_{\rm crit}$ that a halo of mass `haloMass` at time `deltaCritical` should be allowed to take.

Code lines: 10

Contained by: file `merger_trees.branching_probability.generalized_Press_Schechter.F90`

interface: `mergertreebranchingprobabilitygnrlzdprssschchtr`

Description: Constructors for the `generalizedPressSchechter` merger tree builder class.

Code lines: 4

Contained by: file `merger_trees.branching_probability.generalized_Press_Schechter.F90`

file: merger_trees.branching_probability.modifier.F90
Description: Contains a module which provides a class that implements core radii for cored cold mode hot halo mass distributions.
Code lines: 39

module: merger_tree_branching_modifiers
Description: Provides a module which provides a class that implements core radii for cored cold mode hot halo mass distributions.
Code lines: 17
Contained by: file merger_trees.branching_probability.modifier.F90
Used by: subroutine galacticus_function_classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; file merger_trees.branching_probability.generalized_Press_Schechter.F90

file: merger_trees.branching_probability.modifier.Parkinson.F90
Description: Implements a merger tree branching probability rate modifier which uses the model of Parkinson et al. [2008].
Code lines: 109

interface: mergertreebranchingprobabilitymodifierparkinson2008
Description: Constructors for the parkinson2008 merger tree branching probability rate class.
Code lines: 4
Contained by: file merger_trees.branching_probability.modifier.Parkinson.F90

function: parkinson2008constructorinternal
Description: Default constructor for the parkinson2008 merger tree branching probability rate class.
Code lines: 12
Contained by: file merger_trees.branching_probability.modifier.Parkinson.F90
Modules used: input_parameters

function: parkinson2008constructorparameters
Description: A constructor for the parkinson2008 merger tree branching probability rate class which builds the object from a parameter set.
Code lines: 36
Contained by: file merger_trees.branching_probability.modifier.Parkinson.F90
Modules used: input_parameters

function: parkinson2008ratemodifier
Description: Returns a modifier for merger tree branching rates using the Parkinson et al. [2008] algorithm.
Code lines: 16
Contained by: file merger_trees.branching_probability.modifier.Parkinson.F90

file: merger_trees.branching_probability.modifier.identity.F90
Description: Implements a merger tree branching probability rate modifier which always returns the identity modifier.
Code lines: 62

function: `identityconstructorparameters`

Description: A constructor for the `identity` merger tree branching probability rate class which builds the object from a parameter set.
Code lines: 11
Contained by: file `merger_trees.branching_probability.modifier.identity.F90`
Modules used: `input_parameters`

function: `identityratemodifier`

Description: Returns a modifier for merger tree branching rates using the Parkinson et al. [2008] algorithm. Return the core radius of the hot halo mass distribution.
Code lines: 10
Contained by: file `merger_trees.branching_probability.modifier.identity.F90`

interface: `mergertreebranchingprobabilitymodifieridentity`

Description: Constructors for the `identity` merger tree branching probabiliy rate modifier class.
Code lines: 3
Contained by: file `merger_trees.branching_probability.modifier.identity.F90`

file: `merger_trees.construct.F90`

Description: Contains a module which constructs/destructs merger trees.
Code lines: 41

module: `merger_tree_construction`

Description: Constructs/destructs merger trees.
Code lines: 19
Contained by: file `merger_trees.construct.F90`
Modules used: `galacticus_nodes`, `iso_c_binding`
Used by: subroutine `galacticus_function_classes_destroy`, subroutine `galacticus_state_retrieve`, subroutine `galacticus_state_store`, file `tasks.evolve_forests.F90`

file: `merger_trees.construct.build.F90`

Description: Implements a merger tree constructor class which builds merger trees after drawing masses at random from a mass distribution.
Code lines: 314
Modules used: `cosmology_functions`, `cosmology_parameters`, `halo_mass_functions`, `merger_trees_build_masses`, `merger_trees_builders`, `output_times`

function: `buildconstruct`

Description: Build a merger tree.
Code lines: 81
Contained by: file `merger_trees.construct.build.F90`
Modules used: `functions_global`, `galacticus_nodes`, `kind_numbers`, `merger_tree_state_store`, `pseudo_random`, `string_handling`

subroutine: `buildconstructmasses`

Description: Construct the set of tree masses to be built.
Code lines: 42
Contained by: file merger_trees.construct.build.F90
Modules used: galacticus_error memory_management sort

function: buildconstructorinternal
Description: Initializes the merger tree building module.
Code lines: 31
Contained by: file merger_trees.construct.build.F90
Modules used: iso_c_binding numerical_comparison

function: buildconstructorparameters
Description: Constructor for the augment merger tree operator class which takes a parameter set as input.
Code lines: 65
Contained by: file merger_trees.construct.build.F90
Modules used: galacticus_error input_parameters memory_management

subroutine: builddestructor
Description: Destructor for the build merger tree constructor class.
Code lines: 12
Contained by: file merger_trees.construct.build.F90

interface: mergertreeconstructorbuild
Description: Constructors for the build merger tree constructor class.
Code lines: 4
Contained by: file merger_trees.construct.build.F90

file: merger_trees.construct.build.mass_resolution.F90
Description: Contains a module which provides a class of merger tree mass resolutions.
Code lines: 40

module: merger_trees_build_mass_resolution
Description: Provides a class of merger tree mass resolutions.
Code lines: 18
Contained by: file merger_trees.construct.build.mass_resolution.F90
Modules used: galacticus_nodes
Used by: subroutine galacticus_function_classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; file merger_trees.construct.builder.Cole2000.F90

file: merger_trees.construct.build.mass_resolution.fixed.F90
Description: An implementation of a merger tree builder mass resolution which assumes a fixed resolution.
Code lines: 81

function: fixedconstructorinternal
Description: Internal constructor for the fixed merger tree building mass resolution class.

Code lines: 8
Contained by: file merger_trees.construct.build.mass_resolution.fixed.F90

function: fixedconstructorparameters
Description: Constructor for the fixed merger tree building mass resolution class which reads parameters from a provided parameter list.
Code lines: 19
Contained by: file merger_trees.construct.build.mass_resolution.fixed.F90

function: fixedresolution
Description: Returns a fixed mass resolution to use when building merger trees.
Code lines: 9
Contained by: file merger_trees.construct.build.mass_resolution.fixed.F90

interface: mergertreemassresolutionfixed
Description: Constructors for the fixed merger tree resolution class.
Code lines: 4
Contained by: file merger_trees.construct.build.mass_resolution.fixed.F90

file: merger_trees.construct.build.mass_resolution.scaled.F90
Description: An implementation of a merger tree builder mass resolution which assumes a resolution which scales with tree mass.
Code lines: 100

interface: mergertreemassresolutionscaled
Description: Constructors for the scaled merger tree resolution class.
Code lines: 4
Contained by: file merger_trees.construct.build.mass_resolution.scaled.F90

function: scaledconstructorinternal
Description: Internal constructor for the scaled merger tree building mass resolution class.
Code lines: 8
Contained by: file merger_trees.construct.build.mass_resolution.scaled.F90

function: scaledconstructorparameters
Description: Constructor for the scaled merger tree building mass resolution class which reads parameters from a provided parameter list.
Code lines: 36
Contained by: file merger_trees.construct.build.mass_resolution.scaled.F90

function: scaledresolution
Description: Returns a scaled mass resolution to use when building merger trees.
Code lines: 11
Contained by: file merger_trees.construct.build.mass_resolution.scaled.F90
Modules used: galacticus_nodes

file: merger_trees.construct.build.masses.F90
Description: Contains a module which implements a class for creating sets of tree masses to use when building merger trees.
Code lines: 40

module: merger_trees_build_masses
Description: Implements a class for creating sets of tree masses to use when building merger trees.
Code lines: 18
Contained by: file merger_trees.construct.build.masses.F90
Used by: subroutine galacticus_function_classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; file merger_trees.construct.build.F90

file: merger_trees.construct.build.masses.distribution.F90
Description: Contains a module which implements a class providing mass distributions for merger trees to be built.
Code lines: 39

module: merger_trees_build_masses_distributions
Description: Implements a class providing mass distributions for merger trees to be built.
Code lines: 17
Contained by: file merger_trees.construct.build.masses.distribution.F90
Used by: subroutine galacticus_function_classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; file merger_trees.construct.build.masses.sampled_distribution.F90

file: merger_trees.construct.build.masses.distribution.gaussian.F90
Description: Implementation of a merger tree halo mass function sampling class in which the sampling rate is given by a Gaussian distribution in halo mass.
Code lines: 91

function: gaussianconstructorinternal
Description: Internal constructor for the gaussian merger tree halo mass function sampling class.
Code lines: 8
Contained by: file merger_trees.construct.build.masses.distribution.gaussian.F90

function: gaussianconstructorparameters
Description: Constructor for the gaussian merger tree halo mass function sampling class which builds the object from a parameter set.
Code lines: 25
Contained by: file merger_trees.construct.build.masses.distribution.gaussian.F90
Modules used: input_parameters

function: gaussiansample
Description: Computes the halo mass function sampling rate using a volume-limited sampling.
Code lines: 13
Contained by: file merger_trees.construct.build.masses.distribution.gaussian.F90

interface: mergertreebuildmassdistributiongaussian
Description: Constructors for the gaussian merger tree halo mass function sampling class.
Code lines: 4

Contained by: file merger_trees.construct.build.masses.distribution.gaussian.F90

file: merger_trees.construct.build.masses.distribution.halo_mass_function.F90
Description: Implementation of a merger tree halo mass function sampling class in which the sampling rate is proportional to the halo mass function.
Code lines: 148
Modules used: cosmological_density_field halo_mass_functions

function: halomassfunctionconstructorinternal
Description: Internal constructor for the haloMassFunction merger tree halo mass function sampling class.
Code lines: 10
Contained by: file merger_trees.construct.build.masses.distribution.halo_mass_function.F90

function: halomassfunctionconstructorparameters
Description: Constructor for the haloMassFunction merger tree halo mass function sampling class which builds the object from a parameter set.
Code lines: 49
Contained by: file merger_trees.construct.build.masses.distribution.halo_mass_function.F90
Modules used: input_parameters

subroutine: halomassfunctiondestructor
Description: Destructor for the haloMassFunction merger tree halo mass sampling class.
Code lines: 8
Contained by: file merger_trees.construct.build.masses.distribution.halo_mass_function.F90

function: halomassfunctionsample
Description: Computes the halo mass function sampling rate using a volume-limited sampling.
Code lines: 28
Contained by: file merger_trees.construct.build.masses.distribution.halo_mass_function.F90
Modules used: galacticus_nodes

interface: mergertreebuildmassdistributionhalomassfunction
Description: Constructors for the haloMassFunction merger tree halo mass function sampling class.
Code lines: 4
Contained by: file merger_trees.construct.build.masses.distribution.halo_mass_function.F90

file: merger_trees.construct.build.masses.distribution.power_law.F90
Description: Implementation of a merger tree halo mass function sampling class in which the sampling rate is given by a power-law in halo mass.
Code lines: 86

interface: mergertreebuildmassdistributionpowerlaw
Description: Constructors for the powerLaw merger tree halo mass function sampling class.
Code lines: 4
Contained by: file merger_trees.construct.build.masses.distribution.power_law.F90

function: powerlawconstructorinternal
Description: Internal constructor for the powerLaw merger tree halo mass function sampling class.
Code lines: 8

Contained by: file merger_trees.construct.build.masses.distribution.power_law.F90

function: powerlawconstructorparameters
Description: Constructor for the powerLaw merger tree halo mass function sampling class which builds the object from a parameter set.
Code lines: 19
Contained by: file merger_trees.construct.build.masses.distribution.power_law.F90
Modules used: input_parameters

function: powerlawsample
Description: Computes the halo mass function sampling rate using a volume-limited sampling.
Code lines: 14
Contained by: file merger_trees.construct.build.masses.distribution.power_law.F90

file: merger_trees.construct.build.masses.distribution.stellar_mass_function.F90
Description: Implementation of a merger tree halo mass function sampling class optimized to minimize variance in the model stellar mass function.
Code lines: 202
Modules used: conditional_mass_functions halo_mass_functions meta_tree_compute_times

interface: mergertreebuildmassdistributionstllrmssfnctn
Description: Constructors for the stellarMassFunction merger tree halo mass function sampling class.
Code lines: 4
Contained by: file merger_trees.construct.build.masses.distribution.stellar_mass_function.F90

function: stellarmassfunctionconstructorinternal
Description: Internal constructor for the stellarMassFunction merger tree halo mass function sampling class.
Code lines: 11
Contained by: file merger_trees.construct.build.masses.distribution.stellar_mass_function.F90

function: stellarmassfunctionconstructorparameters
Description: Constructor for the stellarMassFunction merger tree halo mass function sampling class which builds the object from a parameter set.
Code lines: 76
Contained by: file merger_trees.construct.build.masses.distribution.stellar_mass_function.F90
Modules used: input_parameters

subroutine: stellarmassfunctiondestructor
Description: Destructor for the stellarMassFunction merger tree halo mass sampling class.
Code lines: 9
Contained by: file merger_trees.construct.build.masses.distribution.stellar_mass_function.F90

function: stellarmassfunctionsample
Description: Computes the halo mass function sampling rate optimized to minimize errors in the stellar mass function.
Code lines: 51
Contained by: file merger_trees.construct.build.masses.distribution.stellar_mass_function.F90
Modules used: fgsl numerical_integration

function: `xiintegrand`
Description: The integrand appearing in the ξ function.
Code lines: 18
Contained by: function `stellarmassfunctionsample`

file: `merger_trees.construct.build.masses.fixed_mass.F90`
Description: Implementation of a merger tree masses class which uses a fixed mass for trees.
Code lines: 215
Modules used: `cosmology_parameters` `dark_matter_halo_scales`

subroutine: `fixedmassconstruct`
Description: Construct a set of merger tree masses by sampling from a distribution.
Code lines: 70
Contained by: file `merger_trees.construct.build.masses.fixed_mass.F90`
Modules used: `galacticus_calculations_resets` `galacticus_nodes` `memory_management` `root_finder` `sort`

function: `massenclosed`
Description: Root finding function used to set the halo mass given the halo radius.
Code lines: 11
Contained by: subroutine `fixedmassconstruct`
Modules used: `galactic_structure_enclosed_masses` `galactic_structure_options`

function: `fixedmassconstructorinternal`
Description: Internal constructor for the `fixedMass` merger tree masses class.
Code lines: 12
Contained by: file `merger_trees.construct.build.masses.fixed_mass.F90`

function: `fixedmassconstructorparameters`
Description: Constructor for the `fixedMass` merger tree masses class which takes a parameter set as input.
Code lines: 72
Contained by: file `merger_trees.construct.build.masses.fixed_mass.F90`
Modules used: `galacticus_error` `input_parameters` `memory_management`

subroutine: `fixedmassdestructor`
Description: Destructor for the `fixedMass` merger tree masses class.
Code lines: 8
Contained by: file `merger_trees.construct.build.masses.fixed_mass.F90`

interface: `mergertreebuildmassesfixedmass`
Code lines: 3
Contained by: file `merger_trees.construct.build.masses.fixed_mass.F90`

file: `merger_trees.construct.build.masses.read.F90`
Description: Implementation of a merger tree masses class which reads masses from a file.

Code lines: 82

type: `mergertreebuildmassesread`
Description: Implementation of a merger tree masses class which reads masses from a file.
Code lines: 17
Contained by: file merger_trees.construct.build.masses.read.F90

subroutine: `readconstruct`
Description: Construct a set of merger tree masses by reading from a file.
Code lines: 28
Contained by: file merger_trees.construct.build.masses.read.F90
Modules used: memory_management sort

file: `merger_trees.construct.build.masses.read.HDF5.F90`
Description: Implementation of a merger tree masses class which reads masses from an HDF5 file.
Code lines: 97

interface: `mergertreebuildmassesreadhdf5`
Code lines: 3
Contained by: file merger_trees.construct.build.masses.read.HDF5.F90

function: `readhdf5constructorinternal`
Description: Internal constructor for the `readHDF5` merger tree masses class.
Code lines: 9
Contained by: file merger_trees.construct.build.masses.read.HDF5.F90

function: `readhdf5constructorparameters`
Description: Constructor for the `readHDF5` merger tree masses class which takes a parameter set as input.
Code lines: 28
Contained by: file merger_trees.construct.build.masses.read.HDF5.F90
Modules used: input_parameters

subroutine: `readhdf5read`
Description: Read merger tree masses from file.
Code lines: 16
Contained by: file merger_trees.construct.build.masses.read.HDF5.F90
Modules used: galacticus_error io_hdf5

file: `merger_trees.construct.build.masses.read.XML.F90`
Description: Implementation of a merger tree masses class which reads masses from an XML file.
Code lines: 104

interface: `mergertreebuildmassesreadxml`
Code lines: 3
Contained by: file merger_trees.construct.build.masses.read.XML.F90

function: `readxmlconstructorinternal`
Description: Internal constructor for the `readXML` merger tree masses class.
Code lines: 9

Contained by: file merger_trees.construct.build.masses.read.XML.F90

function: readxmlconstructorparameters
Description: Constructor for the readXML merger tree masses class which takes a parameter set as input.
Code lines: 28
Contained by: file merger_trees.construct.build.masses.read.XML.F90
Modules used: input_parameters

subroutine: readxmlread
Description: Read merger tree masses from file.
Code lines: 23
Contained by: file merger_trees.construct.build.masses.read.XML.F90
Modules used: fox_dom galacticus_error io_xml

file: merger_trees.construct.build.masses.sampled_distribution.F90
Description: Implementation of a merger tree masses class which samples masses from a distribution.
Code lines: 236
Modules used: merger_trees_build_masses_-distributions

interface: mergertreebuildmassessampleddistribution
Code lines: 2
Contained by: file merger_trees.construct.build.masses.sampled_distribution.F90

subroutine: sampleddistributioncmf
Description: Stub function for cumulative mass function.
Code lines: 10
Contained by: file merger_trees.construct.build.masses.sampled_distribution.F90
Modules used: galacticus_error

subroutine: sampleddistributionconstruct
Description: Construct a set of merger tree masses by sampling from a distribution.
Code lines: 110
Contained by: file merger_trees.construct.build.masses.sampled_distribution.F90
Modules used: fgsl galacticus_error iso_c_binding memory_management numerical_integration numerical_interpolation numerical_ranges sort table_labels

function: distributionintegrand
Description: The integrand over the mass function sampling density function.
Code lines: 7
Contained by: subroutine sampleddistributionconstruct

function: sampleddistributionconstructorparameters
Description: Constructor for the sampledDistribution merger tree masses class which takes a parameter set as input.
Code lines: 43

Contained by: file merger_trees.construct.build.masses.sampled_distribution.F90
Modules used: galacticus_display galacticus_error input_parameters

subroutine: sampleddistributiondestructor
Description: Destructor for the sampledDistribution merger tree masses class.
Code lines: 7
Contained by: file merger_trees.construct.build.masses.sampled_distribution.F90

file: merger_trees.construct.build.masses.sampled_distribution.pseudo_random.F90
Description: Implementation of a merger tree masses class which samples masses from a distribution using pseudo-random sampling.
Code lines: 88

interface: mergertreebuildmassessampleddistributionpseudorandom
Code lines: 3
Contained by: file merger_trees.construct.build.masses.sampled_distribution.pseudo_random.F90

function: sampleddistributionpseudorandomconstructorinternal
Description: Internal constructor for the sampledDistributionPseudoRandom merger tree masses class.
Code lines: 9
Contained by: file merger_trees.construct.build.masses.sampled_distribution.pseudo_random.F90

function: sampleddistributionpseudorandomconstructorparameters
Description: Constructor for the sampledDistributionPseudoRandom merger tree masses class which takes a parameter set as input.
Code lines: 10
Contained by: file merger_trees.construct.build.masses.sampled_distribution.pseudo_random.F90
Modules used: input_parameters

subroutine: sampleddistributionpseudorandomdestructor
Description: Destructor for the sampledDistributionPseudoRandom merger tree masses class.
Code lines: 7
Contained by: file merger_trees.construct.build.masses.sampled_distribution.pseudo_random.F90

subroutine: sampleddistributionpseudorandomsamplecmf
Description: Generate a pseudoRandom sample of points from the merger tree mass distribution.
Code lines: 16
Contained by: file merger_trees.construct.build.masses.sampled_distribution.pseudo_random.F90
Modules used: iso_c_binding pseudo_random

file: merger_trees.construct.build.masses.sampled_distribution.quasi_random.F90
Description: Implementation of a merger tree masses class which samples masses from a distribution using quasi-random sampling.
Code lines: 91

interface: mergertreebuildmassessampleddistributionquasirandom
Code lines: 3
Contained by: file merger_trees.construct.build.masses.sampled_distribution.quasi_random.F90

function: `sampleddistributionquasirandomconstructorinternal`
Description: Internal constructor for the `sampledDistributionQuasiRandom` merger tree masses class.
Code lines: 9
Contained by: file merger_trees.construct.build.masses.sampled_distribution.quasi_random.F90

function: `sampleddistributionquasirandomconstructorparameters`
Description: Constructor for the `sampledDistributionQuasiRandom` merger tree masses class which takes a parameter set as input.
Code lines: 10
Contained by: file merger_trees.construct.build.masses.sampled_distribution.quasi_random.F90
Modules used: input_parameters

subroutine: `sampleddistributionquasirandomdestructor`
Description: Destructor for the `sampledDistributionQuasiRandom` merger tree masses class.
Code lines: 7
Contained by: file merger_trees.construct.build.masses.sampled_distribution.quasi_random.F90

subroutine: `sampleddistributionquasirandomsamplecmf`
Description: Generate a quasiRandom sample of points from the merger tree mass distribution.
Code lines: 19
Contained by: file merger_trees.construct.build.masses.sampled_distribution.quasi_random.F90
Modules used: `fgsl` `iso_c_binding` quasi_random

file: `merger_trees.construct.build.masses.sampled_distribution.uniform.F90`
Description: Implementation of a merger tree masses class which samples masses from a distribution uniformly.
Code lines: 82

interface: `mergertreebuildmassessampleddistributionuniform`
Code lines: 3
Contained by: file merger_trees.construct.build.masses.sampled_distribution.uniform.F90

function: `sampleddistributionuniformconstructorinternal`
Description: Internal constructor for the `sampledDistributionUniform` merger tree masses class.
Code lines: 9
Contained by: file merger_trees.construct.build.masses.sampled_distribution.uniform.F90

function: `sampleddistributionuniformconstructorparameters`
Description: Constructor for the `sampledDistributionUniform` merger tree masses class which takes a parameter set as input.
Code lines: 10
Contained by: file merger_trees.construct.build.masses.sampled_distribution.uniform.F90
Modules used: input_parameters

subroutine: `sampleddistributionuniformdestructor`
Description: Destructor for the `sampledDistributionUniform` merger tree masses class.
Code lines: 7

Contained by: file merger_trees.construct.build.masses.sampled_distribution.uniform.F90

subroutine: sampleddistributionuniformsamplecmf
Description: Generate a uniform sample of points from the merger tree mass distribution.
Code lines: 10
Contained by: file merger_trees.construct.build.masses.sampled_distribution.uniform.F90
Modules used: numerical_ranges

file: merger_trees.construct.build.masses.union.F90
Description: Implementation of a merger tree masses class which constructs the union of other classes.
Code lines: 174

type: mergertreebuildmasseslist
Code lines: 3
Contained by: file merger_trees.construct.build.masses.union.F90

interface: mergertreebuildmassesunion
Code lines: 3
Contained by: file merger_trees.construct.build.masses.union.F90

subroutine: unionconstruct
Description: Construct a set of merger tree masses by sampling from a distribution.
Code lines: 75
Contained by: file merger_trees.construct.build.masses.union.F90
Modules used: galacticus_error, iso_c_binding, memory_management, sort, weight

function: unionconstructorinternal
Description: Internal constructor for the union merger tree masses class.
Code lines: 8
Contained by: file merger_trees.construct.build.masses.union.F90

function: unionconstructorparameters
Description: Constructor for the union merger tree masses class which takes a parameter set as input.
Code lines: 23
Contained by: file merger_trees.construct.build.masses.union.F90
Modules used: input_parameters

subroutine: uniondestructor
Description: Destructor for the merger tree mergerTreeBuildMasses function class.
Code lines: 16
Contained by: file merger_trees.construct.build.masses.union.F90

file: merger_trees.construct.builder.Cole2000.F90
Description: An implementation of a merger tree builder using the algorithm of Cole et al. [2000].
Code lines: 622
Modules used: cosmological_density_field, cosmology_functions, merger_tree_branching, merger_trees_build_mass_resolution

statistics_distributions

subroutine: cole2000build
Description: Build a merger tree.
Code lines: 351
Contained by: file merger_trees.construct.builder.Cole2000.F90
Modules used: galacticus_error, galacticus_nodes, iso_varying_string, kind_numbers, merger_tree_walkers, numerical_comparison, pseudo_random

function: cole2000constructorinternal
Description: Internal constructor for the Cole et al. [2000] merger tree building class.
Code lines: 19
Contained by: file merger_trees.construct.builder.Cole2000.F90
Modules used: galacticus_error

function: cole2000constructorparameters
Description: Constructor for the Cole et al. [2000] merger tree building class which reads parameters from a provided parameter list.
Code lines: 72
Contained by: file merger_trees.construct.builder.Cole2000.F90

subroutine: cole2000criticaloverdensityset
Description: Set the critical overdensity object for this tree builder.
Code lines: 11
Contained by: file merger_trees.construct.builder.Cole2000.F90

function: cole2000criticaloverdensityupdate
Description: Update the critical overdensity for a new node, given that of the parent,
Code lines: 19
Contained by: file merger_trees.construct.builder.Cole2000.F90
Modules used: galacticus_nodes

subroutine: cole2000destructor
Description: Destructor for the cole2000 merger tree builder class.
Code lines: 10
Contained by: file merger_trees.construct.builder.Cole2000.F90

function: cole2000shouldabort
Description: Return true if tree construction should be aborted. In the cole2000 tree builder we never abort.
Code lines: 10
Contained by: file merger_trees.construct.builder.Cole2000.F90

function: cole2000shouldfollowbranch
Description: Return true if tree construction should continue to follow the current branch. In the cole2000 tree builder we always continue.
Code lines: 12
Contained by: file merger_trees.construct.builder.Cole2000.F90
Modules used: galacticus_nodes

subroutine: `cole2000timeearliestset`
Description: Set the earliest time for the tree builder.
Code lines: 8
Contained by: file merger_trees.construct.builder.Cole2000.F90

interface: `mergertreebuildercole2000`
Description: Constructors for the `cole2000` merger tree builder class.
Code lines: 4
Contained by: file merger_trees.construct.builder.Cole2000.F90

file: `merger_trees.construct.builder.F90`
Description: Contains a module which provides a class of merger tree builders.
Code lines: 46

module: `merger_trees_builders`
Description: Provides a class of merger tree builders.
Code lines: 24
Contained by: file merger_trees.construct.builder.F90
Modules used: galacticus_nodes
Used by:

subroutine galacticus_function_-classes_destroy	subroutine galacticus_state_retrieve
subroutine galacticus_state_store	file merger_trees.construct.build.F90
file merger_trees.operators.augment.F90	function augmentaccepttree

file: `merger_trees.construct.fully_specified.F90`
Description: Implements a merger tree constructor class which constructs a merger tree given a full specification in XML.
Code lines: 275
Modules used: fox_dom

type: `documentcontainer`
Description: A container for XML docoment.
Code lines: 5
Contained by: file merger_trees.construct.fully_specified.F90

function: `fullyspecifiedconstruct`
Description: Construct a fully-specified merger tree.
Code lines: 156
Contained by: file merger_trees.construct.fully_specified.F90
Modules used:

fox_dom	galacticus_display
galacticus_error	galacticus_nodes
iso_c_binding	kind_numbers
memory_management	pseudo_random

function: `indexnode`
Description: Extract and return an index from a node definition as used when constructing fully-specified merger trees.
Code lines: 33
Contained by: function fullyspecifiedconstruct

Modules used: fox_dom galacticus_error kind_numbers

function: nodelookup
Description: Find the position of a node in the nodeArray array given its indexValue.
Code lines: 21
Contained by: function fullyspecifiedconstruct
Modules used: galacticus_error galacticus_nodes kind_numbers

function: fullyspecifiedconstructorinternal
Description: Internal constructor for the fullySpecified merger tree operator class.
Code lines: 22
Contained by: file merger_trees.construct.fully_specified.F90
Modules used: galacticus_error

function: fullyspecifiedconstructorparameters
Description: Constructor for the fullySpecified merger tree operator class which takes a parameter set as input.
Code lines: 18
Contained by: file merger_trees.construct.fully_specified.F90
Modules used: input_parameters

subroutine: fullyspecifieddestructor
Description: Destructor for the fullySpecified merger tree constructor class.
Code lines: 16
Contained by: file merger_trees.construct.fully_specified.F90

interface: mergertreeconstructorfullyspecified
Description: Constructors for the fullySpecified merger tree constructor class.
Code lines: 4
Contained by: file merger_trees.construct.fully_specified.F90

file: merger_trees.construct.read.F90
Description: Implements a merger tree constructor class which constructs trees by reading their definitions from file.
Code lines: 3681
Modules used: cosmology_functions dark_matter_halo_scales dark_matter_profile_scales dark_matter_profiles_concentration dark_matter_profiles_dmo galacticus_nodes halo_spin_distributions kind_numbers merger_tree_read_importers mpi_utilities output_times satellite_merging_timescales virial_orbits

interface: mergertreeconstructorread
Description: Constructors for the read merger tree constructor class.
Code lines: 4
Contained by: file merger_trees.construct.read.F90

type: `progenitoriterator`
Code lines: 44
Contained by: file `merger_trees.construct.read.F90`

function: `progenitoriteratorcurrent`
Description: Return a pointer to the current progenitor in a progenitor iterator object.
Code lines: 9
Contained by: file `merger_trees.construct.read.F90`

subroutine: `progenitoriteratordescendentset`
Description: Initialize a progenitor iterator object by storing the index of the taget `node` and finding the location of the first progenitor (if any).
Code lines: 26
Contained by: file `merger_trees.construct.read.F90`

function: `progenitoriteratorexist`
Description: Return true if progenitors exist, false otherwise.
Code lines: 7
Contained by: file `merger_trees.construct.read.F90`

function: `progenitoriteratorindex`
Description: Return the node index of the current progenitor in a progenitor iterator object.
Code lines: 9
Contained by: file `merger_trees.construct.read.F90`

function: `progenitoriteratornext`
Description: Move to the next progenitor using a progenitor iterator object, returning true if the next progenitor exists, false if it does not.
Code lines: 26
Contained by: file `merger_trees.construct.read.F90`

subroutine: `readassignhosttreepointers`
Description: After tree base nodes have been assigned, walk each tree and set the host tree pointer for each node.
Code lines: 18
Contained by: file `merger_trees.construct.read.F90`
Modules used: `merger_tree_walkers`

subroutine: `readassignisolatednodeindices`
Description: Assign to each node the number of the corresponding isolated node.
Code lines: 40
Contained by: file `merger_trees.construct.read.F90`

subroutine: `readassignmergers`
Description: Assign pointers to merge targets.
Code lines: 43
Contained by: file `merger_trees.construct.read.F90`
Modules used: `galacticus_error` `galacticus_nodes`

string_handling

subroutine: `readassignnamedproperties`
Description: Assign named properties to nodes.
Code lines: 37
Contained by: file merger_trees.construct.read.F90
Modules used: galacticus_error galacticus_nodes

subroutine: `readassignscaleradii`
Description: Assign scale radii to nodes.
Code lines: 112
Contained by: file merger_trees.construct.read.F90
Modules used: fallbackscalemethod galacticus_display galacticus_error galacticus_nodes input_parameters root_finder

subroutine: `readassignspinparameters`
Description: Assign spin parameters to nodes.
Code lines: 59
Contained by: file merger_trees.construct.read.F90
Modules used: galacticus_error galacticus_nodes numerical_constants_physical

function: `spinnormalization`
Description: Normalization for conversion of angular momentum to spin.
Code lines: 5
Contained by: subroutine readassignspinparameters

subroutine: `readassignsplitforestevents`
Description: Assign events to nodes if they jump between trees in a forest.
Code lines: 197
Contained by: file merger_trees.construct.read.F90
Modules used: galacticus_display galacticus_error galacticus_nodes node_events_inter_tree string_handling

subroutine: `readassignuniqueidstoclones`
Description: Assign new uniqueID values to any cloned nodes inserted into the trees.
Code lines: 13
Contained by: file merger_trees.construct.read.F90
Modules used: galacticus_nodes

subroutine: `readbuildchildandsiblinglinks`
Description: Build child and sibling links between nodes.
Code lines: 63
Contained by: file merger_trees.construct.read.F90
Modules used: galacticus_nodes memory_management

subroutine: `readbuilddescendentpointers`

Description: Builds pointers from each node to its descendent node.
Code lines: 46
Contained by: file `merger_trees.construct.read.F90`
Modules used: `galacticus_display` `galacticus_error` `string_handling`

subroutine: `readbuildisolatedparentpointers`

Description: Create parent pointer links between isolated nodes and assign times and masses to those nodes.
Code lines: 92
Contained by: file `merger_trees.construct.read.F90`
Modules used: `galacticus_error` `galacticus_nodes` `pseudo_random` `string_handling`

subroutine: `readbuildparentpointers`

Description: Build pointers to node parents.
Code lines: 42
Contained by: file `merger_trees.construct.read.F90`
Modules used: `galacticus_error` `string_handling`

subroutine: `readbuildsubhalomasshistories`

Description: Build and attached bound mass histories to subhalos.
Code lines: 134
Contained by: file `merger_trees.construct.read.F90`
Modules used: `galacticus_error` `galacticus_nodes` `histories` `string_handling`

function: `readconstruct`

Description: Construct a merger tree by reading its definition from file.
Code lines: 313
Contained by: file `merger_trees.construct.read.F90`
Modules used: `array_utilities` `arrays_search` `functions_global` `galacticus_error` `galacticus_nodes` `memory_management` `merger_tree_state_store` `numerical_comparison` `pseudo_random` `sort` `string_handling` `vectors`

function: `readconstructorinternal`

Description: Internal constructor for the `read` merger tree constructor class.
Code lines: 145
Contained by: file `merger_trees.construct.read.F90`
Modules used: `galacticus_display` `galacticus_error` `galacticus_nodes` `memory_management` `numerical_constants_astronomical` `numerical_constants_boolean`

function: `readconstructorparameters`

Description: Constructor for the `read` merger tree constructor class which takes a parameter set as input.
Code lines: 282
Contained by: file `merger_trees.construct.read.F90`
Modules used: `input_parameters`

subroutine: `readcreatebranchjumpevent`
Description: Create a matched-pair of branch jump events in the given nodes.
Code lines: 21
Contained by: file `merger_trees.construct.read.F90`
Modules used: `galacticus_nodes` `node_branch_jumps`

subroutine: `readcreatenodearray`
Description: Create an array of standard nodes and associated structures.
Code lines: 55
Contained by: file `merger_trees.construct.read.F90`
Modules used: `galacticus_nodes` `memory_management`

subroutine: `readcreatenodeindices`
Description: Create a sorted list of node indices with an index into the original array.
Code lines: 31
Contained by: file `merger_trees.construct.read.F90`
Modules used: `galacticus_error` `memory_management` `sort` `string_handling`

function: `readdescendentnodesortindex`
Description: Return the sort index of the given `descendentIndex`.
Code lines: 10
Contained by: file `merger_trees.construct.read.F90`
Modules used: `arrays_search`

subroutine: `readdestroynodeindices`
Description: Destroy the sorted list of node indices.
Code lines: 11
Contained by: file `merger_trees.construct.read.F90`
Modules used: `memory_management`

subroutine: `readdestructor`
Description: Destructor for the `read` merger tree constructor class.
Code lines: 20
Contained by: file `merger_trees.construct.read.F90`

subroutine: `readdumptree`
Description: Dumps the tree structure to a file in a format suitable for processing with DOT.
Code lines: 85
Contained by: file `merger_trees.construct.read.F90`

subroutine: `readenforcesubhalostatus`
Description: Ensure that any node which was once a subhalo remains a subhalo.
Code lines: 69

Contained by: file merger_trees.construct.read.F90
Modules used: galacticus_error string_handling

subroutine: readintertreemergetimeset
Description: Set the merging time for a node undergoing and inter-tree transfer.
Code lines: 79
Contained by: file merger_trees.construct.read.F90
Modules used: galacticus_error galacticus_nodes kepler_orbits string_handling

function: readisonpulllist
Description: Return true if the given node is on the current "pull-from" list of nodes for split forests.
Code lines: 12
Contained by: file merger_trees.construct.read.F90

function: readisonpushlist
Description: Return true if the given node is on the current "push-to" list of nodes for split forests.
Code lines: 12
Contained by: file merger_trees.construct.read.F90

function: readissubhalosubhalomerger
Description: Returns true if node undergoes a subhalo-subhalo merger.
Code lines: 25
Contained by: file merger_trees.construct.read.F90

function: readlasthostdescendent
Description: Return a pointer to the last descendent that can be reached from node when descending through hosts.
Code lines: 11
Contained by: file merger_trees.construct.read.F90

function: readnodelocation
Description: Return the location in the original array of the given nodeIndex.
Code lines: 16
Contained by: file merger_trees.construct.read.F90
Modules used: arrays_search

function: readorbitconstruct
Description: Construct a Keplerian orbit given body masses, positions, and relative velocities.
Code lines: 15
Contained by: file merger_trees.construct.read.F90
Modules used: kepler_orbits vectors

subroutine: readphasespacepositionrealize
Description: Modify relative positions and velocities to account for both any periodicity of the simulated volume, and for Hubble flow.
Code lines: 21
Contained by: file merger_trees.construct.read.F90
Modules used: cosmology_functions numerical_constants_boolean

function: `readpulllistcount`
Description: Return the number of the given node in the "pull-from" list of nodes for split forests.
Code lines: 13
Contained by: file `merger_trees.construct.read.F90`

function: `readpulllistindex`
Description: Return the index of the given node in the "pull-from" list of nodes for split forests.
Code lines: 21
Contained by: file `merger_trees.construct.read.F90`

function: `readpushlistindex`
Description: Return the index of the given node in the "push-to" list of nodes for split forests.
Code lines: 16
Contained by: file `merger_trees.construct.read.F90`

function: `readradiushalfmassroot`
Description: Function used to find scale radius of dark matter halos given their half-mass radius.
Code lines: 10
Contained by: file `merger_trees.construct.read.F90`

subroutine: `readrootnodeaffinitiesinitial`
Description: Find initial root node affinities for all nodes.
Code lines: 205
Contained by: file `merger_trees.construct.read.F90`
Modules used: `galacticus_display` `iso_varying_string` `memory_management` `sort` `string_handling`

subroutine: `readscanforbranchjumps`
Description: Search for subhalos which move between branches/trees.
Code lines: 169
Contained by: file `merger_trees.construct.read.F90`
Modules used: `galacticus_display` `galacticus_nodes` `iso_varying_string` `string_handling`

subroutine: `readscanformergers`
Description: Scan for and record mergers between nodes.
Code lines: 282
Contained by: file `merger_trees.construct.read.F90`
Modules used: `galacticus_error` `galacticus_nodes` `kepler_orbits` `string_handling` `vectors`

subroutine: `readscanforsubhalopromotions`
Description: Scan for cases where a subhalo stops being a subhalo and so must be promoted.
Code lines: 63
Contained by: file `merger_trees.construct.read.F90`
Modules used: `galacticus_nodes` `node_subhalo_promotions`

subroutine: `readtimeuntilmergingsubresolution`
Description: Compute the additional time until merging after a subhalo is lost from the tree (presumably due to limited resolution).
Code lines: 121
Contained by: file merger_trees.construct.read.F90
Modules used: galacticus_display, galacticus_error, galacticus_nodes, kepler_orbits, string_handling, vectors

subroutine: `readtimingrecord`
Description: Record timing data.
Code lines: 26
Contained by: file merger_trees.construct.read.F90

subroutine: `readtimingreport`
Description: Report on time taken in various steps of processing merger trees read from file.
Code lines: 20
Contained by: file merger_trees.construct.read.F90
Modules used: galacticus_display

subroutine: `readvalidateisolatedhalos`
Description: Ensure that nodes have valid primary progenitors.
Code lines: 46
Contained by: file merger_trees.construct.read.F90

file: `merger_trees.construct.read.importer.F90`
Description: Contains a module which provides an object that implements importing of merger trees from file.
Code lines: 377

module: `merger_tree_read_importers`
Description: Provides an object that implements importing of merger trees from file.
Code lines: 355
Contained by: file merger_trees.construct.read.importer.F90
Modules used: galacticus_nodes, iso_c_binding, iso_varying_string, kind_numbers, pseudo_random
Used by: subroutine galacticus_function_classes_destroy, subroutine galacticus_state_retrieve, subroutine galacticus_state_store, file merger_trees.construct.read.F90

interface: `importerunitconvert`
Description: Unit convertors for merger tree importers.
Code lines: 5
Contained by: module merger_tree_read_importers

function: `importerunitconvert1d`
Description: Convert a set of values for Galacticus internal units.

Code lines: 22
Contained by: module merger_tree_read_importers
Modules used: cosmology_functions cosmology_parameters galacticus_error

function: importerunitconvert2d
Description: Convert a set of values for GALACTICUS internal units.
Code lines: 23
Contained by: module merger_tree_read_importers
Modules used: cosmology_functions cosmology_parameters galacticus_error

function: importerunitconvertscalar
Description: Convert a set of values for GALACTICUS internal units.
Code lines: 19
Contained by: module merger_tree_read_importers
Modules used: cosmology_functions cosmology_parameters galacticus_error

type: importerunits
Code lines: 40
Contained by: module merger_tree_read_importers

function: importerunitsareequal
Description: Test whether two importerUnits objects are equal.
Code lines: 10
Contained by: module merger_tree_read_importers
Modules used: galacticus_error

function: importerunitsarenotequal
Description: Test whether two importerUnits objects are not equal.
Code lines: 8
Contained by: module merger_tree_read_importers

function: importerunitsexponentiate
Description: Exponentiate importerUnits objects.
Code lines: 12
Contained by: module merger_tree_read_importers

function: importerunitsmultiply
Description: Multiply to importerUnits objects.
Code lines: 12
Contained by: module merger_tree_read_importers

type: nodedata
Description: Structure used to store default raw data read from merger tree files.
Code lines: 10
Contained by: module merger_tree_read_importers

type: `nodedataminimal`
Description: Structure used to store minimal raw data read from merger tree files.
Code lines: 4
Contained by: module `merger_tree_read_importers`

file: `merger_trees.construct.read.importer.SussingMergerTrees.ASCII.F90`
Description: An implementation of the merger tree importer class for "Sussing Merger Trees" format merger tree files.
Code lines: 880

interface: `mergertreeimportersussingascii`
Description: Constructors for the `sussing` ASCII format merger tree importer class.
Code lines: 4
Contained by: file `merger_trees.construct.read.importer.SussingMergerTrees.ASCII.F90`

function: `sussingasciiconstructorinternal`
Description: Internal constructor for the "Sussing Merger Trees" ASCII format [Srisawat et al., 2013] merger tree importer.
Code lines: 16
Contained by: file `merger_trees.construct.read.importer.SussingMergerTrees.ASCII.F90`

function: `sussingasciiconstructorparameters`
Description: Constructor for the "Sussing Merger Trees" ASCII format [Srisawat et al., 2013] merger tree importer which takes a parameter set as input.
Code lines: 66
Contained by: file `merger_trees.construct.read.importer.SussingMergerTrees.ASCII.F90`
Modules used: `input_parameters`

subroutine: `sussingasciiload`
Description: Load a `sussing` ASCII format merger tree data.
Code lines: 544
Contained by: file `merger_trees.construct.read.importer.SussingMergerTrees.ASCII.F90`
Modules used:

`array_utilities`	`arrays_search`
`file_utilities`	`galacticus_display`
`galacticus_error`	`iso_c_binding`
`kind_numbers`	`memory_management`
`numerical_constants_astronomical`	`numerical_constants_prefixes`
`sort`	`string_handling`

subroutine: `sussingasciiopen`
Description: Validate a `sussing` ASCII format merger tree file.
Code lines: 117
Contained by: file `merger_trees.construct.read.importer.SussingMergerTrees.ASCII.F90`
Modules used:

`file_utilities`	`galacticus_display`
`galacticus_error`	`memory_management`
`numerical_comparison`	`numerical_constants_astronomical`
`regular_expressions`	`string_handling`

subroutine: `sussingasciireadhalo`
Description: Read an ASCII halo definition.
Code lines: 76
Contained by: file `merger_trees.construct.read.importer.SussingMergerTrees.ASCII.F90`
Modules used: `galacticus_error`

file: `merger_trees.construct.read.importer.SussingMergerTrees.F90`
Description: An implementation of the merger tree importer class for "Sussing Merger Trees" format merger tree files.
Code lines: 1033
Modules used: `cosmology_functions` `cosmology_parameters` `iso_varying_string` `pseudo_random` `stateful_types`

interface: `mergertreeimportersussing`
Code lines: 3
Contained by: file `merger_trees.construct.read.importer.SussingMergerTrees.F90`

function: `sussingangularmomenta3davailable`
Description: Return true if angular momenta vectors are available.
Code lines: 8
Contained by: file `merger_trees.construct.read.importer.SussingMergerTrees.F90`

function: `sussingangularmomentaavailable`
Description: Return true if angular momenta are available.
Code lines: 8
Contained by: file `merger_trees.construct.read.importer.SussingMergerTrees.F90`

function: `sussingangularmomentaincludesubhalos`
Description: Return a Boolean specifying whether or not the halo angular momenta include the contribution from subhalos.
Code lines: 9
Contained by: file `merger_trees.construct.read.importer.SussingMergerTrees.F90`
Modules used: `galacticus_error`

function: `sussingcanreadsubsets`
Description: Return false since this format does not permit reading of arbitrary subsets of halos from a forest.
Code lines: 8
Contained by: file `merger_trees.construct.read.importer.SussingMergerTrees.F90`

subroutine: `sussingclose`
Description: Close a `sussing` format merger tree file.
Code lines: 7
Contained by: file `merger_trees.construct.read.importer.SussingMergerTrees.F90`

function: `sussingconstructorinternal`
Description: Internal constructor for the "Sussing Merger Trees" format [Srisawat et al., 2013] merger tree importer class.

Code lines: 16
Contained by: file merger_trees.construct.read.importer.SussingMergerTrees.F90

function: sussingconstructorparameters
Description: Constructor for the "Sussing Merger Trees" format [Srisawat et al., 2013] merger tree importer which takes a parameter set as input.
Code lines: 95
Contained by: file merger_trees.construct.read.importer.SussingMergerTrees.F90
Modules used: input_parameters

function: sussingcubelength
Description: Return the length of the simulation cube.
Code lines: 12
Contained by: file merger_trees.construct.read.importer.SussingMergerTrees.F90
Modules used: numerical_constants_astronomical numerical_constants_boolean

subroutine: sussingdestructor
Code lines: 7
Contained by: file merger_trees.construct.read.importer.SussingMergerTrees.F90

subroutine: sussingimport
Description: Import the $i^{\rm th}$ merger tree.
Code lines: 31
Contained by: file merger_trees.construct.read.importer.SussingMergerTrees.F90
Modules used: galacticus_error memory_management

function: sussinginsubvolume
Description: Determine if a point lies within a subvolume of the simulation box (possibly with some buffering).
Code lines: 9
Contained by: file merger_trees.construct.read.importer.SussingMergerTrees.F90

function: sussinginsubvolume1d
Description: Determine if a point lies within the 1-D range of a subvolume of the simulation box (possibly with some buffering).
Code lines: 23
Contained by: file merger_trees.construct.read.importer.SussingMergerTrees.F90
Modules used: numerical_constants_astronomical

subroutine: sussingload
Description: Stub function for the load method of the sussing merger tree importer.
Code lines: 15
Contained by: file merger_trees.construct.read.importer.SussingMergerTrees.F90
Modules used: galacticus_error

function: sussingmassesincludesubhalos
Description: Return a Boolean specifying whether or not the halo masses include the contribution from subhalos.
Code lines: 9
Contained by: file merger_trees.construct.read.importer.SussingMergerTrees.F90

Modules used: galacticus_error

function: sussingnodecount
Description: Return a count of the number of nodes in the $i^{\rm th}$ tree.
Code lines: 10
Contained by: file merger_trees.construct.read.importer.SussingMergerTrees.F90

function: sussingparticlecountavailable
Description: Return true if particle counts are available.
Code lines: 8
Contained by: file merger_trees.construct.read.importer.SussingMergerTrees.F90

function: sussingperiodicseparation
Description: Determine the separation between two points in a periodic cube.
Code lines: 13
Contained by: file merger_trees.construct.read.importer.SussingMergerTrees.F90

function: sussingpositionsareperiodic
Description: Return a Boolean integer specifying whether or not positions are periodic.
Code lines: 9
Contained by: file merger_trees.construct.read.importer.SussingMergerTrees.F90
Modules used: numerical_constants_boolean

function: sussingpositionsavailable
Description: Return true if positions and/or velocities are available.
Code lines: 9
Contained by: file merger_trees.construct.read.importer.SussingMergerTrees.F90

function: sussingscaleradiiavailable
Description: Return true if scale radii are available.
Code lines: 8
Contained by: file merger_trees.construct.read.importer.SussingMergerTrees.F90

function: sussingspin3davailable
Description: Return true if spins vectors are available.
Code lines: 8
Contained by: file merger_trees.construct.read.importer.SussingMergerTrees.F90

function: sussingspinavailable
Description: Return true if spins are available.
Code lines: 8
Contained by: file merger_trees.construct.read.importer.SussingMergerTrees.F90

subroutine: sussingsubhalotrace
Description: Returns a trace of subhalo position/velocity.
Code lines: 12
Contained by: file merger_trees.construct.read.importer.SussingMergerTrees.F90
Modules used: galacticus_error

function: `sussingsubhalotracecount`
Description: Returns the length of a subhalo trace.
Code lines: 11
Contained by: file merger_trees.construct.read.importer.SussingMergerTrees.F90

function: `sussingtreecount`
Description: Return a count of the number of trees available.
Code lines: 8
Contained by: file merger_trees.construct.read.importer.SussingMergerTrees.F90

function: `sussingtreeindex`
Description: Return the index of the $i^{\rm th}$ tree.
Code lines: 9
Contained by: file merger_trees.construct.read.importer.SussingMergerTrees.F90

subroutine: `sussingtreeindicesread`
Description: Read the tree indices.
Code lines: 381
Contained by: file merger_trees.construct.read.importer.SussingMergerTrees.F90
Modules used:

array_utilities	arrays_search
galacticus_display	galacticus_error
iso_c_binding	kind_numbers
memory_management	numerical_constants_astronomical
numerical_constants_prefixes	sort
string_handling	

function: `sussingtreesareselfcontained`
Description: Return a Boolean integer specifying whether or not the trees are self-contained.
Code lines: 9
Contained by: file merger_trees.construct.read.importer.SussingMergerTrees.F90
Modules used: numerical_constants_boolean

function: `sussingtreeshavesubhalos`
Description: Return a Boolean integer specifying whether or not the trees have subhalos.
Code lines: 9
Contained by: file merger_trees.construct.read.importer.SussingMergerTrees.F90
Modules used: numerical_constants_boolean

function: `sussingtreeweight`
Description: Return the weight to assign to trees.
Code lines: 13
Contained by: file merger_trees.construct.read.importer.SussingMergerTrees.F90
Modules used:

galacticus_error	numerical_constants_astronomical
numerical_constants_boolean	

function: `sussingvalueisbad`
Description: Determine if a value in a “Sussing” merger tree file is bad
Code lines: 17

Contained by: file merger_trees.construct.read.importer.SussingMergerTrees.F90
Modules used: galacticus_error

function: sussingvelocitiesincludehubbleflow
Description: Return a Boolean integer specifying whether or not velocities include the Hubble flow.
Code lines: 9
Contained by: file merger_trees.construct.read.importer.SussingMergerTrees.F90
Modules used: numerical_constants_boolean

function: sussingvelocitydispersionavailable
Description: Return true if halo velocity dispersions are available.
Code lines: 8
Contained by: file merger_trees.construct.read.importer.SussingMergerTrees.F90

function: sussingvelocitymaximumavailable
Description: Return true if halo rotation curve velocity maxima are available.
Code lines: 8
Contained by: file merger_trees.construct.read.importer.SussingMergerTrees.F90

file: merger_trees.construct.read.importer.SussingMergerTrees.HDF5.F90
Description: An implementation of the merger tree importer class for "Sussing Merger Trees" format merger tree files.
Code lines: 474
Modules used: cosmological_density_field io_hdf5

interface: mergertreeimportersussinghdf5
Description: Constructors for the sussing HDF5 format merger tree importer class.
Code lines: 4
Contained by: file merger_trees.construct.read.importer.SussingMergerTrees.HDF5.F90

function: sussinghdf5constructorinternal
Description: Default constructor for the "Sussing Merger Trees" HDF5 format (Thomas et al.; in prep.) merger tree importer.
Code lines: 16
Contained by: file merger_trees.construct.read.importer.SussingMergerTrees.HDF5.F90
Modules used: input_parameters

function: sussinghdf5constructorparameters
Description: Default constructor for the "Sussing Merger Trees" HDF5 format (Thomas et al.; in prep.) merger tree importer.
Code lines: 12
Contained by: file merger_trees.construct.read.importer.SussingMergerTrees.HDF5.F90
Modules used: input_parameters

subroutine: sussinghdf5destructor
Description: Destructor for the sussing HDF5 format merger tree importer class.
Code lines: 12
Contained by: file merger_trees.construct.read.importer.SussingMergerTrees.HDF5.F90
Modules used: memory_management

subroutine: `sussinghdf5load`

Description: Load a `sussing` HDF5 format merger tree data.
Code lines: 270
Contained by: file merger_trees.construct.read.importer.SussingMergerTrees.HDF5.F90
Modules used: array_utilities, arrays_search, file_utilities, galacticus_display, galacticus_error, iso_c_binding, kind_numbers, memory_management, numerical_constants_astronomical, numerical_constants_prefixes, sort, string_handling

function: `decodeunits`

Description: Decode a textual unit definition and construct an importer units object from it.
Code lines: 34
Contained by: subroutine sussinghdf5load
Modules used: galacticus_display, galacticus_error

subroutine: `sussinghdf5open`

Description: Validate a `sussing` HDF5 format merger tree file.
Code lines: 109
Contained by: file merger_trees.construct.read.importer.SussingMergerTrees.HDF5.F90
Modules used: file_utilities, galacticus_display, galacticus_error, memory_management, numerical_comparison, numerical_constants_astronomical, string_handling

file: `merger_trees.construct.read.importer.galacticus.F90`

Description: An implementation of the merger tree importer class for Galacticus format merger tree files.
Code lines: 1224
Modules used: cosmological_density_field, cosmology_functions, cosmology_parameters, halo_mass_functions, io_hdf5, stateful_types

function: `galacticusangularmomenta3davailable`

Description: Return true if angular momenta vectors are available.
Code lines: 7
Contained by: file merger_trees.construct.read.importer.galacticus.F90

function: `galacticusangularmomentaavailable`

Description: Return true if angular momenta are available.
Code lines: 7
Contained by: file merger_trees.construct.read.importer.galacticus.F90

function: `galacticusangularmomentaincludesubhalos`

Description: Return a Boolean specifying whether or not the halo momenta include the contribution from subhalos.
Code lines: 24
Contained by: file merger_trees.construct.read.importer.galacticus.F90

Modules used: galacticus_error

function: galacticuscanreadsubsets
Description: Return true since this format does permit reading of arbitrary subsets of halos from a forest.
Code lines: 8
Contained by: file merger_trees.construct.read.importer.galacticus.F90

subroutine: galacticusclose
Description: Validate a Galacticus format merger tree file.
Code lines: 11
Contained by: file merger_trees.construct.read.importer.galacticus.F90

function: galacticusconstructorinternal
Description: Internal constructor for the Galacticus format merger tree importer.
Code lines: 19
Contained by: file merger_trees.construct.read.importer.galacticus.F90

function: galacticusconstructorparameters
Description: Constructor for the Galacticus format merger tree importer which takes a parameter set as input.
Code lines: 39
Contained by: file merger_trees.construct.read.importer.galacticus.F90
Modules used: input_parameters

function: galacticuscubelength
Description: Return the length of the simulation cube.
Code lines: 41
Contained by: file merger_trees.construct.read.importer.galacticus.F90
Modules used: galacticus_error numerical_constants_astronomical numerical_constants_boolean

subroutine: galacticusdestructor
Description: Destructor for the Galacticus format merger tree importer class.
Code lines: 14
Contained by: file merger_trees.construct.read.importer.galacticus.F90

subroutine: galacticusforestindicesread
Description: Read the tree indices.
Code lines: 109
Contained by: file merger_trees.construct.read.importer.galacticus.F90
Modules used: galacticus_error hdf5 numerical_constants_astronomical sort

subroutine: galacticusimport
Description: Import the $i^{\rm th}$ merger tree.
Code lines: 303
Contained by: file merger_trees.construct.read.importer.galacticus.F90
Modules used: galacticus_display galacticus_error hdf5 memory_management

nodesubset numerical_constants_astronomical
numerical_constants_prefixes vectors

function: galacticusmassesincludesubhalos
Description: Return a Boolean specifying whether or not the halo masses include the contribution from subhalos.
Code lines: 20
Contained by: file merger_trees.construct.read.importer.galacticus.F90
Modules used: galacticus_error

function: galacticusnodecount
Description: Return a count of the number of nodes in the $i^{\rm th}$ tree.
Code lines: 10
Contained by: file merger_trees.construct.read.importer.galacticus.F90

subroutine: galacticusopen
Description: Validate a Galacticus format merger tree file.
Code lines: 211
Contained by: file merger_trees.construct.read.importer.galacticus.F90
Modules used: galacticus_display galacticus_error
numerical_comparison

function: galacticusparticlecountavailable
Description: Return true if particle counts are available.
Code lines: 9
Contained by: file merger_trees.construct.read.importer.galacticus.F90

function: galacticuspositionsareperiodic
Description: Return a Boolean integer specifying whether or not positions are periodic.
Code lines: 18
Contained by: file merger_trees.construct.read.importer.galacticus.F90
Modules used: numerical_constants_boolean

function: galacticuspositionsavailable
Description: Return true if positions and/or velocities are available.
Code lines: 12
Contained by: file merger_trees.construct.read.importer.galacticus.F90

function: galacticusscaleradiiavailable
Description: Return true if scale radii are available.
Code lines: 9
Contained by: file merger_trees.construct.read.importer.galacticus.F90

function: galacticusspin3davailable
Description: Return true if spins vectors are available.
Code lines: 7
Contained by: file merger_trees.construct.read.importer.galacticus.F90

function: galacticusspinavailable

Description: Return true if spins are available.
Code lines: 7
Contained by: file merger_trees.construct.read.importer.galacticus.F90

subroutine: galacticussubhalotrace
Description: Returns a trace of subhalo position/velocity.
Code lines: 39
Contained by: file merger_trees.construct.read.importer.galacticus.F90
Modules used: galacticus_error numerical_constants_astronomical

function: galacticussubhalotracecount
Description: Returns the length of a subhalo trace.
Code lines: 17
Contained by: file merger_trees.construct.read.importer.galacticus.F90
Modules used: galacticus_error

function: galacticustreecount
Description: Return a count of the number of trees available.
Code lines: 9
Contained by: file merger_trees.construct.read.importer.galacticus.F90

function: galacticustreeindex
Description: Return the index of the $i^{\rm th}$ tree.
Code lines: 9
Contained by: file merger_trees.construct.read.importer.galacticus.F90

function: galacticustreesareselfcontained
Description: Return a Boolean integer specifying whether or not the trees are self-contained.
Code lines: 18
Contained by: file merger_trees.construct.read.importer.galacticus.F90
Modules used: numerical_constants_boolean

function: galacticustreeshavesubhalos
Description: Return a Boolean integer specifying whether or not the trees have subhalos.
Code lines: 18
Contained by: file merger_trees.construct.read.importer.galacticus.F90
Modules used: numerical_constants_boolean

function: galacticustreeweight
Description: Return the weight to assign to trees.
Code lines: 30
Contained by: file merger_trees.construct.read.importer.galacticus.F90
Modules used: galacticus_error numerical_constants_astronomical
numerical_constants_boolean

function: galacticusvelocitiesincludehubbleflow
Description: Return a Boolean integer specifying whether or not velocities include the Hubble flow.
Code lines: 18
Contained by: file merger_trees.construct.read.importer.galacticus.F90

Modules used: numerical_constants_boolean

function: galacticusvelocitydispersionavailable
Description: Return true if halo velocity dispersions are available.
Code lines: 9
Contained by: file merger_trees.construct.read.importer.galacticus.F90

function: galacticusvelocitymaximumavailable
Description: Return true if halo rotation curve velocity maxima are available.
Code lines: 9
Contained by: file merger_trees.construct.read.importer.galacticus.F90

interface: mergertreeimportergalacticus
Description: Constructors for the GALACTICUS format merger tree importer class.
Code lines: 4
Contained by: file merger_trees.construct.read.importer.galacticus.F90

type: nodedatagalacticus
Description: Extension of the nodeData class for GALACTICUS format merger trees. Stores particle indices and counts for nodes.
Code lines: 3
Contained by: file merger_trees.construct.read.importer.galacticus.F90

file: merger_trees.construct.smooth_accretion.F90
Description: Implements a merger tree constructor class which builds merger trees assuming smooth accretion.
Code lines: 192
Modules used: cosmology_functions dark_matter_halo_mass_accretion_histories

interface: mergertreeconstructorsmoothaccretion
Description: Constructors for the smoothAccretion merger tree constructor class.
Code lines: 4
Contained by: file merger_trees.construct.smooth_accretion.F90

function: smoothaccretionconstruct
Description: Build a merger tree with a smooth mass accretion history using the fitting function of Wechsler et al. [2002].
Code lines: 72
Contained by: file merger_trees.construct.smooth_accretion.F90
Modules used: galacticus_nodes iso_c_binding kind_numbers pseudo_random

function: smoothaccretionconstructorinternal
Description: Internal constructor for the augment merger tree operator class.
Code lines: 10
Contained by: file merger_trees.construct.smooth_accretion.F90

function: smoothaccretionconstructorparameters
Description: Constructor for the augment merger tree operator class which takes a parameter set as input.

Code lines: 49
Contained by: file merger_trees.construct.smooth_accretion.F90
Modules used: input_parameters

subroutine: smoothaccretiondestructor
Description: Destructor for the smoothAccretion merger tree constructor class.
Code lines: 8
Contained by: file merger_trees.construct.smooth_accretion.F90

file: merger_trees.construct.state_restore.F90
Description: Implements a merger tree constructor class which constructs a merger tree by restoring state from file.
Code lines: 378

interface: mergertreeconstructorstaterestored
Description: Constructors for the stateRestored merger tree constructor class.
Code lines: 4
Contained by: file merger_trees.construct.state_restore.F90

subroutine: mergertreestatefromfile
Description: Read the state of a merger tree from file.
Code lines: 142
Contained by: file merger_trees.construct.state_restore.F90
Modules used: galacticus_error, galacticus_nodes, kind_numbers, string_handling

function: nodepointer
Description: Return a pointer to a node, given its position in the array of nodes. Return a null pointer if the array index is -1.
Code lines: 13
Contained by: subroutine mergertreestatefromfile
Modules used: galacticus_nodes

subroutine: mergertreestatestore
Description: Store the complete internal state of a merger tree to file.
Code lines: 137
Contained by: file merger_trees.construct.state_restore.F90
Modules used: functions_global, galacticus_error, galacticus_nodes, iso_c_binding, kind_numbers, merger_tree_walkers

function: nodearrayposition
Description: Returns the position of a node in the output list given its index.
Code lines: 25
Contained by: subroutine mergertreestatestore
Modules used: galacticus_error, iso_varying_string, kind_numbers, string_handling

function: staterestoredconstruct

Description: Restores the state of a merger tree from file.
Code lines: 21
Contained by: file merger_trees.construct.state_restore.F90
Modules used: functions_global galacticus_nodes iso_c_binding

function: staterestoredconstructorinternal
Description: Internal constructor for the stateRestored merger tree operator class.
Code lines: 8
Contained by: file merger_trees.construct.state_restore.F90

function: staterestoredconstructorparameters
Description: Constructor for the stateRestored merger tree operator class which takes a parameter set as input.
Code lines: 19
Contained by: file merger_trees.construct.state_restore.F90
Modules used: input_parameters

file: merger_trees.dump_evolution.F90
Description: Contains a module which dumps the evolution of merger trees to XML.
Code lines: 103

module: merger_trees_dump_evolution
Description: Dumps the structure of entire merger trees.
Code lines: 81
Contained by: file merger_trees.dump_evolution.F90
Modules used: iso_varying_string
Used by: subroutine galacticus_output_close_file subroutine standardevolve

subroutine: merger_tree_dump_evolution
Description: Trim histories attached to the disk.
Code lines: 41
Contained by: module merger_trees_dump_evolution
Modules used: galacticus_nodes input_parameters

subroutine: merger_tree_dump_evolution_close
Description: Close the merger tree evolution dump file.
Code lines: 9
Contained by: module merger_trees_dump_evolution

file: merger_trees.dump_structure.F90
Description: Contains a module which dumps the structure of entire merger trees.
Code lines: 119

module: merger_tree_dump_structure
Description: Dumps the structure of entire merger trees.
Code lines: 97
Contained by: file merger_trees.dump_structure.F90
Modules used: iso_varying_string

Used by: subroutine evolveforestsperform

subroutine: merger_tree_structure_dump
Description: Output the structure of thisTree.
Code lines: 71
Contained by: module merger_tree_dump_structure
Modules used: galacticus_nodes input_parameters merger_trees_dump

file: merger_trees.evolve.deadlock_options.F90
Description: Contains a module which provides an enumeration for tree deadlock statuses.
Code lines: 34

module: merger_trees_evolve_deadlock_status
Description: Provides an enumeration for tree deadlock statuses.
Code lines: 12
Contained by: file merger_trees.evolve.deadlock_options.F90
Used by: function node_branch_jump; function node_pull_from_tree; function node_push_from_tree; function node_subhalo_promotion; subroutine satellitemergerprocess; subroutine standardevolve; subroutine standardnodeeventsperform

file: merger_trees.evolve.timesteps.F90
Description: Contains a module which implements a class for merger tree evolution timestepping.
Code lines: 58

module: merger_tree_timesteps
Description: Implements a class for merger tree evolution timestepping.
Code lines: 36
Contained by: file merger_trees.evolve.timesteps.F90
Modules used: galacticus_nodes
Used by: subroutine galacticus_function_-classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; file merger_trees.evolver.standard.F90; subroutine standardevolve; function standardtimeevolveto

file: merger_trees.evolve.timesteps.history.F90
Description: Implements a merger tree evolution timestepping class which limits the step the next epoch at which to store global history.
Code lines: 334
Modules used: cosmology_functions fgsl

subroutine: historyautohook
Description: Create a hook to the HDF5 pre-close event to allow us to finalize and write out our data.
Code lines: 8
Contained by: file merger_trees.evolve.timesteps.history.F90
Modules used: events_hooks

function: `historyconstructorinternal`
Description: Constructor for the `history` merger tree evolution timestep class which takes a parameter set as input.
Code lines: 44
Contained by: file merger_trees.evolve.timesteps.history.F90
Modules used: galacticus_nodes, iso_c_binding, memory_management, numerical_ranges

function: `historyconstructorparameters`
Description: Constructor for the `history` merger tree evolution timestep class which takes a parameter set as input.
Code lines: 43
Contained by: file merger_trees.evolve.timesteps.history.F90
Modules used: input_parameters

subroutine: `historydestructor`
Description: Destructor for the `history` merger tree evolution timestep class.
Code lines: 7
Contained by: file merger_trees.evolve.timesteps.history.F90

subroutine: `historystore`
Description: Store various properties in global arrays.
Code lines: 60
Contained by: file merger_trees.evolve.timesteps.history.F90
Modules used: galactic_structure_enclosed_masses, galactic_structure_options, galacticus_error, galacticus_nodes, iso_c_binding, numerical_interpolation

function: `historytimeevolveto`
Description: Determine a suitable timestep for `node` using the history method.
Code lines: 36
Contained by: file merger_trees.evolve.timesteps.history.F90
Modules used: evolve_to_time_reports, galacticus_nodes, iso_c_binding, iso_varying_string, numerical_interpolation

subroutine: `historywrite`
Description: Store the global history data to the Galacticus output file.
Code lines: 73
Contained by: file merger_trees.evolve.timesteps.history.F90
Modules used: galacticus_error, galacticus_hdf5, numerical_constants_astronomical

interface: `mergertreeevolvetimestephistory`
Description: Constructors for the `history` merger tree evolution timestep class.
Code lines: 4
Contained by: file merger_trees.evolve.timesteps.history.F90

file: `merger_trees.evolve.timesteps.multi.F90`
Description: Implements a class for applying multiple different timestepping criteria.
Code lines: 174

interface: `mergertreeevolvetimestepmulti`
Description: Constructors for the `multi` mergerTreeEvolveTimestep.
Code lines: 4
Contained by: file `merger_trees.evolve.timesteps.multi.F90`

function: `multiconstructorinternal`
Description: Internal constructor for the `multi` merger tree evolution timestep class.
Code lines: 14
Contained by: file `merger_trees.evolve.timesteps.multi.F90`

function: `multiconstructorparameters`
Description: Constructor for the `multi` merger tree evolution timestep class which takes a parameter set as input.
Code lines: 22
Contained by: file `merger_trees.evolve.timesteps.multi.F90`
Modules used: `input_parameters`

subroutine: `multideepcopy`
Description: Perform a deep copy for the `multi` merger tree evolution timestep class.
Code lines: 31
Contained by: file `merger_trees.evolve.timesteps.multi.F90`
Modules used: `galacticus_error`

subroutine: `multidestructor`
Description: Destructor for the `multi` merger tree evolution timestep class.
Code lines: 16
Contained by: file `merger_trees.evolve.timesteps.multi.F90`

type: `multimergertreeevolvetimesteplist`
Code lines: 3
Contained by: file `merger_trees.evolve.timesteps.multi.F90`

function: `multitimeevolveto`
Description: Perform all mergerTreeEvolveTimesteps.
Code lines: 35
Contained by: file `merger_trees.evolve.timesteps.multi.F90`

file: `merger_trees.evolve.timesteps.record_evolution.F90`
Description: Implements a merger tree evolution timestepping class which limits the step to the next epoch at which to record evolution of the main branch galaxy.
Code lines: 287
Modules used: `cosmology_functions` `fgsl` `output_times`

interface: `mergertreeevolvetimesteprecordevolution`

Description: Constructors for the `recordEvolution` merger tree evolution timestep class.
Code lines: 4
Contained by: file `merger_trees.evolve.timesteps.record_evolution.F90`

subroutine: `recordevolutionautohook`
Description: Create a hook to the merger tree extra output event to allow us to write out our data.
Code lines: 8
Contained by: file `merger_trees.evolve.timesteps.record_evolution.F90`
Modules used: `events_hooks`

function: `recordevolutionconstructorinternal`
Description: Internal constructor for the `recordEvolution` merger tree evolution timestep class.
Code lines: 25
Contained by: file `merger_trees.evolve.timesteps.record_evolution.F90`
Modules used: `iso_c_binding` `memory_management` `numerical_ranges`

function: `recordevolutionconstructorparameters`
Description: Constructor for the `recordEvolution` merger tree evolution timestep class which takes a parameter set as input.
Code lines: 43
Contained by: file `merger_trees.evolve.timesteps.record_evolution.F90`
Modules used: `input_parameters`

subroutine: `recordevolutiondestructor`
Description: Destructor for the `recordEvolution` merger tree evolution timestep class.
Code lines: 8
Contained by: file `merger_trees.evolve.timesteps.record_evolution.F90`

subroutine: `recordevolutionoutput`
Description: Store main branch evolution to the output file.
Code lines: 45
Contained by: file `merger_trees.evolve.timesteps.record_evolution.F90`
Modules used: `galacticus_error` `galacticus_hdf5` `iso_c_binding` `iso_varying_string` `numerical_constants_astronomical` `string_handling`

subroutine: `recordevolutionreset`
Description: Resets recorded datasets to zero.
Code lines: 8
Contained by: file `merger_trees.evolve.timesteps.record_evolution.F90`

subroutine: `recordevolutionstore`
Description: Store properties of the main progenitor galaxy.
Code lines: 33
Contained by: file `merger_trees.evolve.timesteps.record_evolution.F90`
Modules used: `galactic_structure_enclosed_masses` `galactic_structure_options` `galacticus_error` `galacticus_nodes` `iso_c_binding` `numerical_interpolation`

function: `recordevolutiontimeevolveto`
Description: Determines the timestep to go to the next tabulation point for galaxy evolution storage.
Code lines: 38
Contained by: file `merger_trees.evolve.timesteps.record_evolution.F90`
Modules used: `evolve_to_time_reports` `galacticus_nodes` `iso_c_binding` `iso_varying_string` `numerical_interpolation`

file: `merger_trees.evolve.timesteps.report.F90`
Description: Contains a module which reports on timestepping criteria.
Code lines: 52

module: `evolve_to_time_reports`
Description: Contains functions which report on timestepping criteria.
Code lines: 30
Contained by: file `merger_trees.evolve.timesteps.report.F90`
Used by: function `historytimeevolveto` function `recordevolutiontimeevolveto` function `satellitetimeevolveto` function `simpletimeevolveto` function `standardtimeevolveto` function `standardtimeevolveto`

subroutine: `evolve_to_time_report`
Description: Display a report on evolution timestep criteria.
Code lines: 20
Contained by: module `evolve_to_time_reports`
Modules used: `galacticus_display` `iso_varying_string` `kind_numbers` `string_handling`

file: `merger_trees.evolve.timesteps.satellite.F90`
Code lines: 247

interface: `mergertreeevolvetimestepsatellite`
Description: Constructors for the `satellite` merger tree evolution timestep class.
Code lines: 4
Contained by: file `merger_trees.evolve.timesteps.satellite.F90`

function: `satelliteconstructorinternal`
Description: Constructor for the `satellite` merger tree evolution timestep class which takes a parameter set as input.
Code lines: 10
Contained by: file `merger_trees.evolve.timesteps.satellite.F90`
Modules used: `galacticus_nodes`

function: `satelliteconstructorparameters`
Description: Constructor for the `satellite` merger tree evolution timestep class which takes a parameter set as input.
Code lines: 29
Contained by: file `merger_trees.evolve.timesteps.satellite.F90`
Modules used: `input_parameters`

subroutine: `satellitemergerprocess`

Description: Process a satellite node which has undergone a merger with its host node.
Code lines: 103
Contained by: file merger_trees.evolve.timesteps.satellite.F90
Modules used: galacticus_display, iso_varying_string, merger_trees_evolve_deadlock_status, node_component_age_statistics_-standard, node_component_black_hole_simple, node_component_black_hole_standard, node_component_disk_standard, node_component_disk_very_simple, node_component_hot_halo_cold_mode, node_component_hot_halo_standard, node_component_hot_halo_very_simple, node_component_hot_halo_vs_delayed, node_component_inter_output_standard, node_component_merging_statistics_-major, node_component_merging_statistics_-standard, node_component_spheroid_standard, node_component_spheroid_very_simple, satellite_merging_remnant_properties, satellites_merging_output, string_handling

function: `satellitetimeevolveto`

Description: Determine a suitable timestep for `node` such that it does not exceed the time of the next satellite merger.
Code lines: 59
Contained by: file merger_trees.evolve.timesteps.satellite.F90
Modules used: evolve_to_time_reports, galacticus_nodes, iso_varying_string

file: `merger_trees.evolve.timesteps.simple.F90`

Code lines: 130
Modules used: cosmology_functions

interface: `mergertreeevolvetimestepsimple`

Description: Constructors for the `simple` merger tree evolution timestep class.
Code lines: 4
Contained by: file merger_trees.evolve.timesteps.simple.F90

function: `simpleconstructorinternal`

Description: Constructor for the `simple` merger tree evolution timestep class which takes a parameter set as input.
Code lines: 9
Contained by: file merger_trees.evolve.timesteps.simple.F90

function: `simpleconstructorparameters`

Description: Constructor for the `simple` merger tree evolution timestep class which takes a parameter set as input.
Code lines: 32
Contained by: file merger_trees.evolve.timesteps.simple.F90
Modules used: input_parameters

subroutine: `simpledestructor`
Description: Destructor for the `simple` merger tree evolution timestep class.
Code lines: 7
Contained by: file `merger_trees.evolve.timesteps.simple.F90`

function: `simpletimeevolveto`
Description: Determine a suitable timestep for `node` using the simple method. This simply selects the smaller of `timeStepAbsolute` and `timeStepRelative`$H^{-1}(t)$.
Code lines: 33
Contained by: file `merger_trees.evolve.timesteps.simple.F90`
Modules used: `evolve_to_time_reports` `galacticus_nodes` `iso_varying_string`

file: `merger_trees.evolve.timesteps.standard.F90`
Code lines: 124
Modules used: `cosmology_functions`

interface: `mergertreeevolvetimestepstandard`
Description: Constructors for the `standard` merger tree evolution timestep class.
Code lines: 4
Contained by: file `merger_trees.evolve.timesteps.standard.F90`

function: `standardconstructorinternal`
Description: Internal constructor for the `standard` merger tree evolution timestep class.
Code lines: 11
Contained by: file `merger_trees.evolve.timesteps.standard.F90`

function: `standardconstructorparameters`
Description: Constructor for the `standard` merger tree evolution timestep class which takes a parameter set as input.
Code lines: 14
Contained by: file `merger_trees.evolve.timesteps.standard.F90`
Modules used: `input_parameters`

subroutine: `standarddestructor`
Description: Destructor for the `standard` merger tree evolution timestep class.
Code lines: 8
Contained by: file `merger_trees.evolve.timesteps.standard.F90`

function: `standardtimeevolveto`
Description: Determine a suitable timestep for `node` by combining the `simple` and `satellite` timesteps.
Code lines: 35
Contained by: file `merger_trees.evolve.timesteps.standard.F90`
Modules used: `evolve_to_time_reports` `iso_varying_string`

file: `merger_trees.evolver.F90`
Description: Contains a module which provides a class that implements evolution of merger trees.
Code lines: 48

module: merger_trees_evolve
Description: Provides a class that implements evolution of merger trees.
Code lines: 26
Contained by: file merger_trees.evolver.F90
Modules used: galacticus_nodes, kind_numbers
Used by: subroutine galacticus_function_-classes_destroy, subroutine galacticus_state_retrieve, subroutine galacticus_state_store, file tasks.evolve_forests.F90

file: merger_trees.evolver.non_evolving.F90
Description: Implements a non-evolving class for evolving merger trees.
Code lines: 81

interface: mergertreeevolvernonevolving
Description: Constructors for the nonEvolving merger tree evolver.
Code lines: 3
Contained by: file merger_trees.evolver.non_evolving.F90

function: nonevolvingconstructorparameters
Description: Constructor for the nonEvolving merger tree evolver class which takes a parameter set as input.
Code lines: 10
Contained by: file merger_trees.evolver.non_evolving.F90
Modules used: input_parameters

subroutine: nonevolvingevolve
Description: Evolves all properties of a merger tree to the specified time.
Code lines: 30
Contained by: file merger_trees.evolver.non_evolving.F90
Modules used: galacticus_error, merger_trees_initialize

file: merger_trees.evolver.standard.F90
Description: Implements the standard class for evolving merger trees.
Code lines: 1029
Modules used: cosmology_functions, galactic_structure_solvers, galacticus_nodes, kind_numbers, merger_tree_timesteps, merger_trees_evolve_node

type: deadlocklist
Code lines: 5
Contained by: file merger_trees.evolver.standard.F90

interface: mergertreeevolverstandard
Description: Constructors for the standard merger tree evolver.
Code lines: 4
Contained by: file merger_trees.evolver.standard.F90

function: `standardconstructorinternal`
Description: Internal constructor for the `standard` merger tree evolver class.
Code lines: 14
Contained by: file `merger_trees.evolver.standard.F90`

function: `standardconstructorparameters`
Description: Constructor for the `standard` merger tree evolver class which takes a parameter set as input.
Code lines: 59
Contained by: file `merger_trees.evolver.standard.F90`
Modules used: `input_parameters`

subroutine: `standarddeadlockaddnode`
Description: Add a node to the deadlocked nodes list.
Code lines: 27
Contained by: file `merger_trees.evolver.standard.F90`

subroutine: `standarddeadlockoutputtree`
Description: Output the deadlocked nodes in `dot` format.
Code lines: 113
Contained by: file `merger_trees.evolver.standard.F90`
Modules used: `galacticus_nodes` `string_handling`

subroutine: `standarddestructor`
Description: Destructor for the `standard`m erger tree evolver class.
Code lines: 10
Contained by: file `merger_trees.evolver.standard.F90`

subroutine: `standardevolve`
Description: Evolves all properties of a merger tree to the specified time.
Code lines: 377
Contained by: file `merger_trees.evolver.standard.F90`
Modules used: `galacticus_display` `galacticus_error` `galacticus_nodes` `input_parameters` `merger_tree_timesteps` `merger_tree_walkers` `merger_trees_dump` `merger_trees_evolve_deadlock_status` `merger_trees_evolve_node` `merger_trees_initialize` `string_handling`

subroutine: `standardnodeeventsperform`
Description: Perform any events associated with `node`.
Code lines: 58
Contained by: file `merger_trees.evolver.standard.F90`
Modules used: `galacticus_nodes` `merger_trees_evolve_deadlock_status`

function: `standardtimeevolveto`
Description: Determine the time to which `node` should be evolved.
Code lines: 226
Contained by: file `merger_trees.evolver.standard.F90`

Modules used: evolve_to_time_reports, galacticus_display, galacticus_error, galacticus_nodes, input_parameters, merger_tree_timesteps, merger_trees_evolve_node, string_handling

subroutine: standardtreeeventsperform
Description: Perform any events associated with tree.
Code lines: 42
Contained by: file merger_trees.evolver.standard.F90
Modules used: galacticus_nodes

file: merger_trees.initialize.F90
Description: Contains a module which implements initialization of merger tree structures.
Code lines: 131

module: merger_trees_initialize
Description: Implements initialization of merger tree structures.
Code lines: 109
Contained by: file merger_trees.initialize.F90
Used by: subroutine nonevolvingevolve, subroutine standardevolve, subroutine evolveforestsperform

subroutine: merger_tree_initialize
Description: Walk through all nodes of a tree and call any routines that requested to perform initialization tasks.
Code lines: 99
Contained by: module merger_trees_initialize
Modules used: galacticus_nodes, merger_tree_walkers, node_component_basic_extended_tracking, node_component_basic_standard, node_component_basic_standard_extended, node_component_basic_standard_tracking, node_component_dark_matter_profile_scale, node_component_dark_matter_profile_scale_preset, node_component_dark_matter_profile_scale_shape, node_component_formation_times_cole2000, node_component_formation_times_mass_fraction, node_component_host_history_standard, node_component_hot_halo_cold_mode, node_component_hot_halo_standard, node_component_hot_halo_very_simple, node_component_hot_halo_vs_delayed, node_component_indices_standard, node_component_mass_flow_statistics_standard, node_component_merging_statistics_recent, node_component_merging_statistics_standard, node_component_position_trace_dark_matter, node_component_satellite_orbiting, node_component_satellite_standard, node_component_satellite_very_simple, node_component_spin_preset, node_component_spin_preset3d

node_component_spin_random node_component_spin_vitvitska

file: merger_trees.node_evolver.F90
Description: Contains a module which provides a class that implements evolution of nodes.
Code lines: 67

module: merger_trees_evolve_node
Description: Provides a class that implements evolution of nodes.
Code lines: 45
Contained by: file merger_trees.node_evolver.F90
Modules used: galactic_structure_solvers, galacticus_nodes
Used by: function node_subhalo_promotion, subroutine galacticus_function_classes_destroy, subroutine galacticus_state_retrieve, subroutine galacticus_state_store, file merger_trees.evolver.standard.F90, subroutine standardevolve, function standardtimeevolveto

file: merger_trees.node_evolver.standard.F90
Description: Implements the standard class for evolving nodes in merger trees.
Code lines: 1524
Modules used: fodeiv2, kind_numbers, merger_trees_merge_node

interface: mergertreenodeevolverstandard
Description: Constructors for the standard merger tree node evolver.
Code lines: 4
Contained by: file merger_trees.node_evolver.standard.F90

function: standardconstructorinternal
Description: Internal constructor for the standard merger tree node evolver class.
Code lines: 59
Contained by: file merger_trees.node_evolver.standard.F90
Modules used: fodeiv2, galacticus_error

function: standardconstructorparameters
Description: Constructor for the standard merger tree node evolver class which takes a parameter set as input.
Code lines: 95
Contained by: file merger_trees.node_evolver.standard.F90
Modules used: input_parameters

subroutine: standardderivativescompute
Description: Call routines to set alls derivatives for node.
Code lines: 110
Contained by: file merger_trees.node_evolver.standard.F90
Modules used: galacticus_calculations_resets, node_component_age_statistics_standard, node_component_basic_non_evolving, node_component_basic_standard, node_component_basic_standard_extended, node_component_black_hole_noncentral

node_component_black_hole_simple, node_component_black_hole_standard, node_component_dark_matter_profile_scale, node_component_dark_matter_profile_scale_preset, node_component_dark_matter_profile_scale_shape, node_component_disk_standard, node_component_disk_very_simple, node_component_dynamics_statistics_bars, node_component_formation_times_cole2000, node_component_hot_halo_cold_mode, node_component_hot_halo_outflow_tracking, node_component_hot_halo_standard, node_component_hot_halo_very_simple, node_component_hot_halo_vs_delayed, node_component_inter_output_standard, node_component_mass_flow_statistics_standard, node_component_satellite_orbiting, node_component_satellite_preset, node_component_satellite_standard, node_component_satellite_very_simple, node_component_spheroid_standard, node_component_spheroid_very_simple, node_component_spin_preset, node_component_spin_preset3d, node_component_spin_vitvitska

subroutine: `standarddestructor`

Description: Destructor for the `standard` merger tree node evolver class.

Code lines: 9

Contained by: file merger_trees.node_evolver.standard.F90

Modules used: odeiv2_solver

subroutine: `standarderrorhandler`

Description: Handles errors in the ODE solver when evolving Galacticus nodes. Dumps the content of the node.

Code lines: 62

Contained by: file merger_trees.node_evolver.standard.F90

Modules used: fodeiv2, galacticus_display, iso_c_binding, string_handling

subroutine: `standardevolve`

Description: Evolves `node` to time `timeEnd`, or until evolution is interrupted.

Code lines: 505

Contained by: file merger_trees.node_evolver.standard.F90

Modules used: galacticus_calculations_resets, galacticus_display, galacticus_error, galacticus_meta_tree_timing, galacticus_nodes, iso_c_binding, memory_management, merger_trees_dump_evolution, node_component_age_statistics_standard, node_component_basic_non_evolving, node_component_basic_standard, node_component_basic_standard_extended, node_component_black_hole_noncentral, node_component_black_hole_simple, node_component_black_hole_standard, node_component_dark_matter_profile_scale

node_component_dark_matter_profile_scale_preset
node_component_dark_matter_profile_scale_shape
node_component_disk_standard
node_component_disk_very_simple
node_component_host_history_standard
node_component_hot_halo_cold_mode
node_component_hot_halo_outflow_tracking
node_component_hot_halo_standard
node_component_hot_halo_very_simple
node_component_hot_halo_vs_delayed
node_component_inter_output_standard
node_component_mass_flow_statistics_standard
node_component_merging_statistics_standard
node_component_satellite_orbiting
node_component_satellite_standard
node_component_spheroid_standard
node_component_spheroid_very_simple
node_component_spin_preset
node_component_spin_preset3d
node_component_spin_vitvitska
numerical_integration2
ode_solver_error_codes
odeiv2_solver

subroutine: `standardfinalstateprocessing`

Description: Perform any actions based on the final state of the ODE step.
Code lines: 34
Contained by: file merger_trees.node_evolver.standard.F90
Modules used: node_component_disk_standard

subroutine: `standardintegrands`

Description: A set of integrands for unit tests.
Code lines: 44
Contained by: file merger_trees.node_evolver.standard.F90
Modules used: galacticus_nodes

function: `standardisaccurate`

Description: Return true if a tree node property is within expected accuracy of a given value.
Code lines: 9
Contained by: file merger_trees.node_evolver.standard.F90
Modules used: numerical_comparison

subroutine: `standardmerge`

Description: Handles instances where `node` is about to merge with its parent node.
Code lines: 71
Contained by: file merger_trees.node_evolver.standard.F90
Modules used: galacticus_display
node_component_basic_standard
node_component_basic_standard_extended
node_component_host_history_standard
node_component_hot_halo_cold_mode
node_component_hot_halo_standard
node_component_hot_halo_very_simple
node_component_hot_halo_vs_delayed
node_component_merging_statistics_recent
node_component_merging_statistics_standard
node_component_position_preset
node_component_position_trace_dark_matter

node_component_satellite_orbiting node_component_satellite_standard
node_component_satellite_very_simple string_handling

function: standardodes
Description: Function which evaluates the set of ODEs for the evolution of a specific node.
Code lines: 78
Contained by: file merger_trees.node_evolver.standard.F90
Modules used: fgsl fodeiv2
galacticus_error galacticus_nodes
ode_solver_error_codes

function: standardodesjacobian
Description: Function which evaluates the set of ODEs for the evolution of a specific node.
Code lines: 58
Contained by: file merger_trees.node_evolver.standard.F90
Modules used: fgsl

function: standardodesteptolerances
Description: Compute the tolerances on each property being evolved in the ODE stystem at the current timestep.
Code lines: 11
Contained by: file merger_trees.node_evolver.standard.F90

subroutine: standardpoststepprocessing
Description: Perform any post-step actions on the node.
Code lines: 49
Contained by: file merger_trees.node_evolver.standard.F90
Modules used: fgsl iso_c_binding
node_component_basic_standard node_component_black_hole_standard
node_component_disk_standard node_component_disk_very_simple
node_component_hot_halo_standard node_component_spheroid_standard
node_component_spheroid_very_simple

subroutine: standardpromote
Description: Transfer the properties of node to its parent node, then destroy it.
Code lines: 159
Contained by: file merger_trees.node_evolver.standard.F90
Modules used: galacticus_display node_component_basic_extended_-
tracking
node_component_basic_non_evolving node_component_basic_standard
node_component_basic_standard_- node_component_basic_standard_-
extended tracking
node_component_dark_matter_profile_- node_component_dark_matter_profile_-
scale scale_preset
node_component_dark_matter_profile_- node_component_formation_times_-
scale_shape cole2000
node_component_formation_times_mass_- node_component_host_history_standard
fraction

node_component_hot_halo_cold_mode node_component_hot_halo_standard
node_component_hot_halo_very_simple node_component_hot_halo_vs_delayed
node_component_merging_statistics_major node_component_merging_statistics_recent
node_component_merging_statistics_standard node_component_nbody_generic
node_component_position_preset node_component_satellite_preset
node_component_spin_preset node_component_spin_preset3d
node_component_spin_random node_component_spin_vitvitska
node_promotion_index_shifts string_handling

subroutine: standardsteperroranalyzer
Description: Profiles ODE solver step sizes and errors.
Code lines: 34
Contained by: file merger_trees.node_evolver.standard.F90
Modules used: fgsl galacticus_meta_evolver_profiler iso_c_binding

file: merger_trees.node_merger.F90
Description: Contains a module which provides a class that implements processing of mergers between nodes.
Code lines: 40

module: merger_trees_merge_node
Description: Provides a class that implements processing of mergers between nodes.
Code lines: 18
Contained by: file merger_trees.node_merger.F90
Modules used: galacticus_nodes
Used by: subroutine galacticus_function_classes_destroy subroutine galacticus_state_retrieve
subroutine galacticus_state_store file merger_trees.node_evolver.standard.F90

file: merger_trees.node_merger.single_level_hierarchy.F90
Description: Provides a node merger class implementing a single level hierarchy.
Code lines: 97

interface: mergertreenodemergersinglelevelhierarchy
Description: Constructors for the singleLevelHierarchy merger tree evolver.
Code lines: 3
Contained by: file merger_trees.node_merger.single_level_hierarchy.F90

function: singlelevelhierarchyconstructorparameters
Description: Constructor for the singleLevelHierarchy merger tree evolver class which takes a parameter set as input.
Code lines: 10
Contained by: file merger_trees.node_merger.single_level_hierarchy.F90
Modules used: input_parameters

subroutine: singlelevelhierarchyprocess

Description: Processes a node merging event, utilizing a single level substructure hierarchy.
Code lines: 45
Contained by: file merger_trees.node_merger.single_level_hierarchy.F90
Modules used: galacticus_error, galacticus_nodes, satellite_promotion, string_handling

file: merger_trees.operators.F90
Code lines: 49

module: merger_tree_operators
Description: Provides an object that implements operators acting on merger trees.
Code lines: 29
Contained by: file merger_trees.operators.F90
Modules used: galacticus_nodes
Used by: subroutine galacticus_function_-classes_destroy, subroutine galacticus_state_retrieve, subroutine galacticus_state_store, file tasks.evolve_forests.F90

file: merger_trees.operators.assign_orbits.F90
Description: Contains a module which implements a merger tree operator which assigns orbits to non-primary progenitor nodes.
Code lines: 251
Modules used: satellite_merging_timescales, virial_orbits

function: assignorbitsconstructorinternal
Description: Constructor for the orbit assigning merger tree operator class which takes a parameter set as input.
Code lines: 10
Contained by: file merger_trees.operators.assign_orbits.F90
Modules used: input_parameters

function: assignorbitsconstructorparameters
Description: Constructor for the orbit assigning merger tree operator class which takes a parameter set as input.
Code lines: 16
Contained by: file merger_trees.operators.assign_orbits.F90
Modules used: input_parameters

subroutine: assignorbitsdestructor
Description: Destructor for the merger tree operator function class.
Code lines: 8
Contained by: file merger_trees.operators.assign_orbits.F90

subroutine: assignorbitsoperate
Description: Perform a orbit assigning operation on a merger tree.
Code lines: 165
Contained by: file merger_trees.operators.assign_orbits.F90
Modules used: galacticus_nodes, kepler_orbits, merger_tree_walkers

interface: `mergertreeoperatorassignorbits`
Description: Constructors for the orbit assigning merger tree operator class.
Code lines: 4
Contained by: file `merger_trees.operators.assign_orbits.F90`

file: `merger_trees.operators.augment.F90`
Description: Contains a module which implements an augmenting operator on merger trees.
Code lines: 1209
Modules used: `cosmology_functions` `iso_c_binding` `merger_trees_builders`

function: `augmentaccepttree`
Description: Determine whether a trial tree is an acceptable match to the original tree structure.
Code lines: 259
Contained by: file `merger_trees.operators.augment.F90`
Modules used: `galacticus_display` `galacticus_nodes` `merger_tree_walkers` `merger_trees_builders`

function: `augmentbuildtreefromnode`
Code lines: 184
Contained by: file `merger_trees.operators.augment.F90`
Modules used: `arrays_search` `galacticus_error` `galacticus_nodes` `iso_c_binding` `numerical_comparison` `string_handling`

function: `augmentchildcount`
Description: Return a count of the number of child nodes.
Code lines: 13
Contained by: file `merger_trees.operators.augment.F90`
Modules used: `galacticus_nodes`

function: `augmentconstructorinternal`
Description: Internal constructor for the `augment` merger tree operator class.
Code lines: 29
Contained by: file `merger_trees.operators.augment.F90`
Modules used: `cosmology_functions` `memory_management` `sort`

function: `augmentconstructorparameters`
Description: Constructor for the `augment` merger tree operator class which takes a parameter set as input.
Code lines: 128
Contained by: file `merger_trees.operators.augment.F90`
Modules used: `galacticus_error` `input_parameters` `memory_management`

subroutine: `augmentdestructor`
Description: Destructor for the augment merger tree operator function class.
Code lines: 8

Contained by: file merger_trees.operators.augment.F90

subroutine: augmentextendbyoverlap

Description: Conjoin two trees by overlapping the nodeTop of one tree with the chosen nodeBottom of the other. If keepTop is true, nodeTop replaces nodeBottom, otherwise, nodeBottom replaces nodeTop. If exchangeProperties is true, the mass and time information of the deleted node overwrites the mass and time of the retained node.
Code lines: 51
Contained by: file merger_trees.operators.augment.F90
Modules used: galacticus_nodes

subroutine: augmentextendnonoverlapnodes

Description: Extend any non-overlap nodes in an accepted tree by growing a new tree from each such node.
Code lines: 26
Contained by: file merger_trees.operators.augment.F90
Modules used: galacticus_error galacticus_nodes

subroutine: augmentfinalize

Description: Output augmentation histogram.
Code lines: 34
Contained by: file merger_trees.operators.augment.F90
Modules used: galacticus_hdf5 io_hdf5
iso_varying_string memory_management

function: augmentnodecomparison

Description: Compare the masses of an overlap node and the corresponding node in the original tree, testing if they agree to withing tolerance.
Code lines: 17
Contained by: file merger_trees.operators.augment.F90
Modules used: galacticus_nodes

subroutine: augmentnonoverlaplistadd

Description: Add the given node to a linked list of non-overlap nodes in the current trial tree.
Code lines: 32
Contained by: file merger_trees.operators.augment.F90
Modules used: galacticus_nodes

subroutine: augmentnonoverlapreinsert

Description: Reinsert non-overlap nodes into their tree.
Code lines: 23
Contained by: file merger_trees.operators.augment.F90
Modules used: galacticus_nodes

subroutine: augmentoperate

Description: Augment the resolution of a merger tree by inserting high resolution branches.
Code lines: 150
Contained by: file merger_trees.operators.augment.F90
Modules used: galacticus_display galacticus_nodes

iso_c_binding merger_tree_walkers
merger_trees_pruning_utilities sort
string_handling

subroutine: augmentsimpleinsert
Description: Insert a newly constructed tree into the original tree.
Code lines: 34
Contained by: file merger_trees.operators.augment.F90
Modules used: galacticus_nodes

subroutine: augmentsortchildren
Description: Sort the children of the given node such that they are in descending mass order.
Code lines: 50
Contained by: file merger_trees.operators.augment.F90
Modules used: galacticus_error galacticus_nodes

function: augmenttreestatistics
Description: Walks through tree and quietly collects information specified by desiredOutput input enumeration and returns that information.
Code lines: 31
Contained by: file merger_trees.operators.augment.F90
Modules used: galacticus_error galacticus_nodes
merger_tree_walkers

interface: mergertreeoperatoraugment
Description: Constructors for the augment merger tree operator class.
Code lines: 4
Contained by: file merger_trees.operators.augment.F90

file: merger_trees.operators.conditional_mass_function.F90
Description: Contains a module which implements a merger tree operator which accumulates conditional mass functions for trees.
Code lines: 1071
Modules used: cosmology_functions statistics_nbody_halo_mass_errors

function: conditionalmfbinweights
Description: Computes the weight that a given halo contributes to an array of bins.
Code lines: 33
Contained by: file merger_trees.operators.conditional_mass_function.F90
Modules used: galacticus_nodes

function: conditionalmfbinweights2d
Description: Computes the weight that a given halo contributes to a 2D array of bins.
Code lines: 102
Contained by: file merger_trees.operators.conditional_mass_function.F90
Modules used: fgsl galacticus_error
galacticus_nodes numerical_integration

function: conditionalmfbinweights2dintegrand

Description: Integrand used in finding the weight given to a bin in the space of parent mass vs. progenitor mass ratio.
Code lines: 26
Contained by: function conditionalmfbinweights2d
Modules used: galacticus_error numerical_constants_math

function: conditionalmfconstructorinternal
Description: Internal constructor for the conditional mass function merger tree operator class.
Code lines: 89
Contained by: file merger_trees.operators.conditional_mass_function.F90
Modules used: galacticus_error galacticus_nodes memory_management numerical_ranges

function: conditionalmfconstructorparameters
Description: Constructor for the conditional mass function merger tree operator class which takes a parameter set as input.
Code lines: 144
Contained by: file merger_trees.operators.conditional_mass_function.F90
Modules used: memory_management

subroutine: conditionalmfdestructor
Description: Destructor for the merger tree operator function class.
Code lines: 10
Contained by: file merger_trees.operators.conditional_mass_function.F90

subroutine: conditionalmffinalize
Description: Outputs conditional mass function.
Code lines: 220
Contained by: file merger_trees.operators.conditional_mass_function.F90
Modules used: galacticus_hdf5 io_hdf5 iso_varying_string memory_management numerical_constants_astronomical

subroutine: conditionalmfoperate
Description: Compute conditional mass function on tree.
Code lines: 264
Contained by: file merger_trees.operators.conditional_mass_function.F90
Modules used: galacticus_error galacticus_nodes input_parameters memory_management merger_tree_walkers numerical_comparison

interface: mergertreeoperatorconditionalmf
Description: Constructors for the conditional mass function merger tree operator class.
Code lines: 4
Contained by: file merger_trees.operators.conditional_mass_function.F90

file: merger_trees.operators.deforest.F90
Description: Contains a module which implements a deforestation operator on merger trees (i.e. removes all but the most massive tree in a forest).
Code lines: 109

function: `deforestconstructorparameters`
Description: Constructor for the deforestation merger tree operator class which takes a parameter set as input.
Code lines: 10
Contained by: file `merger_trees.operators.deforest.F90`
Modules used: `input_parameters`

subroutine: `deforestdestructor`
Description: Destructor for the deforestation merger tree operator function class.
Code lines: 8
Contained by: file `merger_trees.operators.deforest.F90`

subroutine: `deforestoperate`
Description: Perform a deforestation operation on a merger tree.
Code lines: 46
Contained by: file `merger_trees.operators.deforest.F90`
Modules used: `galacticus_nodes`

interface: `mergertreeoperatordeforest`
Description: Constructors for the deforestation merger tree operator class.
Code lines: 3
Contained by: file `merger_trees.operators.deforest.F90`

file: `merger_trees.operators.dump_to_GraphViz.F90`
Description: Contains a module which implements a dump to GraphViz operator on merger trees.
Code lines: 143

function: `dumptographvizconstructorinternal`
Description: Internal constructor for the dump-to-GraphViz merger tree operator class.
Code lines: 11
Contained by: file `merger_trees.operators.dump_to_GraphViz.F90`

function: `dumptographvizconstructorparameters`
Description: Constructor for the dump-to-GraphViz merger tree operator class which takes a parameter set as input.
Code lines: 54
Contained by: file `merger_trees.operators.dump_to_GraphViz.F90`
Modules used: `input_parameters`

subroutine: `dumptographvizdestructor`
Description: Destructor for the merger tree operator function class.
Code lines: 8
Contained by: file `merger_trees.operators.dump_to_GraphViz.F90`

subroutine: `dumptographvizoperate`
Description: Output the structure of `tree`.
Code lines: 20
Contained by: file `merger_trees.operators.dump_to_GraphViz.F90`

Modules used: galacticus_nodes merger_trees_dump

interface: mergertreeoperatordumptographviz
Description: Constructors for the dump to GraphViz merger tree operator class.
Code lines: 4
Contained by: file merger_trees.operators.dump_to_GraphViz.F90

file: merger_trees.operators.export.F90
Description: Contains a module which implements a merger tree operator which exports merger trees to file.
Code lines: 285
Modules used: cosmological_density_field cosmology_functions cosmology_parameters

function: exportconstructorinternal
Description: Internal constructor for the export merger tree operator class.
Code lines: 18
Contained by: file merger_trees.operators.export.F90
Modules used: galacticus_error merger_tree_data_structure

function: exportconstructorparameters
Description: Constructor for the export merger tree operator class which takes a parameter set as input.
Code lines: 41
Contained by: file merger_trees.operators.export.F90
Modules used: input_parameters merger_tree_data_structure

subroutine: exportdestructor
Description: Destructor for the export merger tree operator function class.
Code lines: 9
Contained by: file merger_trees.operators.export.F90

subroutine: exportoperate
Description: Output the structure of thisTree.
Code lines: 157
Contained by: file merger_trees.operators.export.F90
Modules used: dates_and_times fgsl
file_utilities galacticus_nodes
hdf5 input_parameters
memory_management merger_tree_data_structure
merger_tree_walkers numerical_constants_astronomical
numerical_interpolation sort
system_command

interface: mergertreeoperatorexport
Description: Constructors for the export merger tree operator class.
Code lines: 4
Contained by: file merger_trees.operators.export.F90

file: merger_trees.operators.information_content.F90

Description: Contains a module which implements a merger tree operator which computes the cladistic information content Thorley et al. [1998] of merger trees.
Code lines: 195
Modules used: `iso_c_binding` `kind_numbers`

function: `informationcontentconstructorinternal`
Description: Internal constructor for the information content merger tree operator class.
Code lines: 10
Contained by: file `merger_trees.operators.information_content.F90`
Modules used: `galacticus_error`

function: `informationcontentconstructorparameters`
Description: Constructor for the information content merger tree operator class which takes a parameter set as input.
Code lines: 18
Contained by: file `merger_trees.operators.information_content.F90`

subroutine: `informationcontentdestructor`
Description: Destructor for the merger tree operator function class.
Code lines: 8
Contained by: file `merger_trees.operators.information_content.F90`

subroutine: `informationcontentfinalize`
Description: Outputs tree information content function.
Code lines: 25
Contained by: file `merger_trees.operators.information_content.F90`
Modules used: `galacticus_hdf5` `hdf5` `io_hdf5`

subroutine: `informationcontentoperate`
Description: Perform a information content operation on a merger tree.
Code lines: 68
Contained by: file `merger_trees.operators.information_content.F90`
Modules used: `factorials` `memory_management` `merger_tree_walkers` `merger_trees_pruning_utilities`

interface: `mergertreeoperatorinformationcontent`
Description: Constructors for the prune-hierarchy merger tree operator class.
Code lines: 4
Contained by: file `merger_trees.operators.information_content.F90`

file: `merger_trees.operators.mass_accretion_history.F90`
Description: Contains a module which implements a merger tree operator which outputs mass accretion histories.
Code lines: 229
Modules used: `cosmology_functions` `io_hdf5`

function: `massaccretionhistoryconstructorinternal`
Description: Internal constructor for the mass accretion history merger tree operator class.
Code lines: 14

Contained by: file merger_trees.operators.mass_accretion_history.F90
Modules used: galacticus_error galacticus_nodes

function: massaccretionhistoryconstructorparameters
Description: Constructor for the mass accretion history merger tree operator class which takes a parameter set as input.
Code lines: 42
Contained by: file merger_trees.operators.mass_accretion_history.F90

subroutine: massaccretionhistorydestructor
Description: Destructor for the mass accretion history merger tree operator function class.
Code lines: 7
Contained by: file merger_trees.operators.mass_accretion_history.F90

subroutine: massaccretionhistoryfinalize
Description: Close the mass accretion history group before closing the HDF5 file.
Code lines: 9
Contained by: file merger_trees.operators.mass_accretion_history.F90

subroutine: massaccretionhistoryoperate
Description: Output the mass accretion history for a merger tree.
Code lines: 90
Contained by: file merger_trees.operators.mass_accretion_history.F90
Modules used: galacticus_error galacticus_hdf5 galacticus_nodes input_parameters iso_c_binding iso_varying_string memory_management numerical_constants_astronomical string_handling

interface: mergertreeoperatormassaccretionhistory
Description: Constructors for the mass accretion history merger tree operator class.
Code lines: 4
Contained by: file merger_trees.operators.mass_accretion_history.F90

file: merger_trees.operators.monotonize_mass_growth.F90
Description: Contains a module which implements a merger tree operator which makes mass growth along branch monotonically increasing.
Code lines: 114

interface: mergertreeoperatormonotonizemassgrowth
Description: Constructors for the mass growth monotonizing merger tree operator class.
Code lines: 3
Contained by: file merger_trees.operators.monotonize_mass_growth.F90

function: monotonizemassgrowthconstructorparameters
Description: Constructor for the mass growth monotonizing merger tree operator class which takes a parameter set as input.
Code lines: 11
Contained by: file merger_trees.operators.monotonize_mass_growth.F90
Modules used: input_parameters

subroutine: `monotonizemassgrowthdestructor`
Description: Destructor for the mass growth monotonizing merger tree operator function class.
Code lines: 8
Contained by: file `merger_trees.operators.monotonize_mass_growth.F90`

subroutine: `monotonizemassgrowthoperate`
Description: Perform a mass growth monotonizing operation on a merger tree.
Code lines: 47
Contained by: file `merger_trees.operators.monotonize_mass_growth.F90`
Modules used: `galacticus_nodes` `merger_tree_walkers`

file: `merger_trees.operators.null.F90`
Description: Contains a module which implements a null operator on merger trees.
Code lines: 71

interface: `mergertreeoperatornull`
Description: Constructors for the null merger tree operator class.
Code lines: 3
Contained by: file `merger_trees.operators.null.F90`

function: `nullconstructorparameters`
Description: Constructor for the null merger tree operator class which takes a parameter set as input.
Code lines: 10
Contained by: file `merger_trees.operators.null.F90`
Modules used: `input_parameters`

subroutine: `nulldestructor`
Description: Destructor for the merger tree operator function class.
Code lines: 8
Contained by: file `merger_trees.operators.null.F90`

subroutine: `nulloperate`
Description: Perform a null operation on a merger tree.
Code lines: 9
Contained by: file `merger_trees.operators.null.F90`

file: `merger_trees.operators.output_root_masses.F90`
Description: Contains a module which implements a merger tree operator which outputs a file of tree root masses (and weights).
Code lines: 204

interface: `mergertreeoperatoroutputrootmasses`
Description: Constructors for the tree root mass outputting merger tree operator class.
Code lines: 4
Contained by: file `merger_trees.operators.output_root_masses.F90`

function: `outputrootmassesconstructorinternal`
Description: Internal constructor for the conditional mass function merger tree operator class.

Code lines: 18
Contained by: file merger_trees.operators.output_root_masses.F90
Modules used: file_utilities system_command

function: outputrootmassesconstructorparameters
Description: Constructor for the conditional mass function merger tree operator class which takes a parameter set as input.
Code lines: 42
Contained by: file merger_trees.operators.output_root_masses.F90
Modules used: cosmology_functions

subroutine: outputrootmassesfinalize
Description: Outputs conditional mass function.
Code lines: 23
Contained by: file merger_trees.operators.output_root_masses.F90
Modules used: galacticus_hdf5 io_hdf5
iso_varying_string

subroutine: outputrootmassesoperate
Description: Compute conditional mass function on tree.
Code lines: 66
Contained by: file merger_trees.operators.output_root_masses.F90
Modules used: galacticus_error galacticus_nodes
merger_tree_walkers numerical_comparison

file: merger_trees.operators.particulate.F90
Description: Contains a module which implements a merger tree operator which creates particle representations of Galacticus halos.
Code lines: 841
Modules used: cosmology_functions cosmology_parameters
dark_matter_halo_scales dark_matter_profiles_dmo
galacticus_nodes hdf5
input_parameters iso_varying_string
kind_numbers tables

interface: mergertreeoperatorparticulate
Description: Constructors for the particulate merger tree operator class.
Code lines: 4
Contained by: file merger_trees.operators.particulate.F90

function: particulateconstructorinternal
Description: Internal constructor for the particulate merger tree operator class.
Code lines: 21
Contained by: file merger_trees.operators.particulate.F90
Modules used: galacticus_error

function: particulateconstructorparameters
Description: Constructor for the particulate merger tree operator class which takes a parameter set as input.
Code lines: 173

Contained by: file merger_trees.operators.particulate.F90
Modules used: input_parameters

subroutine: particulatedestructor
Description: Destructor for the particulate merger tree operator class.
Code lines: 10
Contained by: file merger_trees.operators.particulate.F90

function: particulateeddingtonintegrand
Description: The integrand appearing in Eddington's formula for the distribution function.
Code lines: 14
Contained by: file merger_trees.operators.particulate.F90

function: particulatemassintegrand
Description: The integrand used to find the enclosed mass in the smoothed density profile defined by Barnes [2012] to account for gravitational softening.
Code lines: 13
Contained by: file merger_trees.operators.particulate.F90
Modules used: numerical_constants_math

subroutine: particulateoperate
Description: Perform a particulation operation on a merger tree (i.e. create a particle representation of the tree).
Code lines: 294
Contained by: file merger_trees.operators.particulate.F90
Modules used:
coordinates
galactic_structure_enclosed_masses
galactic_structure_options
galacticus_calculations_resets
galacticus_display
galacticus_error
galacticus_nodes
io_hdf5
iso_varying_string
memory_management
merger_tree_walkers
node_components
numerical_comparison
numerical_constants_math
pseudo_random

function: particulatepotentialintegrand
Description: The integrand used to find the gravitational potential in the smoothed density profile defined by Barnes [2012] to account for gravitational softening.
Code lines: 11
Contained by: file merger_trees.operators.particulate.F90
Modules used: numerical_constants_physical

function: particulatesmoothingintegrandr
Description: The integrand over cylindrical coordinate z used in finding the smoothed density profile defined by Barnes [2012] to account for gravitational softening.
Code lines: 25
Contained by: file merger_trees.operators.particulate.F90

function: particulatesmoothingintegrandz
Description: The integrand over cylindrical coordinate z used in finding the smoothed density profile defined by Barnes [2012] to account for gravitational softening.

Code lines: 26
Contained by: file merger_trees.operators.particulate.F90
Modules used: fgsl numerical_integration

subroutine: particulatetabulateenergydistribution
Description: Construct the energy distribution function assuming a spherical dark matter halo with isotropic velocity dispersion. We solve Eddington's formula [Binney and Tremaine, 2008, eqn. 4.43a].

$$f(E) = \frac{1}{\sqrt{8}\pi^2} \frac{\mathrm{d}}{\mathrm{d}E} \int_E^0 \frac{\mathrm{d}\Phi}{\sqrt{\Phi - E}} \frac{\mathrm{d}\rho}{\mathrm{d}\Phi}. \tag{18.22}$$

In practice, we tabulate:

$$F(E) = \int_E^0 \frac{\mathrm{d}\Phi}{\sqrt{\Phi - E}} \frac{\mathrm{d}\rho}{\mathrm{d}\Phi}, \tag{18.23}$$

which we can then take the derivative of numerically to obtain the distribution function.
Code lines: 131
Contained by: file merger_trees.operators.particulate.F90
Modules used: fgsl galacticus_error numerical_integration table_labels

file: merger_trees.operators.perturb_masses.F90
Description: Contains a module which implements a merger tree operator which perturbs halo masses by some error model.
Code lines: 171
Modules used: statistics_distributions statistics_nbody_halo_mass_errors

interface: mergertreeoperatorperturbmasses
Description: Constructors for the mass perturbing merger tree operator class.
Code lines: 4
Contained by: file merger_trees.operators.perturb_masses.F90

function: perturbmassesconstructorinternal
Description: Constructor for the mass perturbing merger tree operator class which takes a parameter set as input.
Code lines: 12
Contained by: file merger_trees.operators.perturb_masses.F90
Modules used: input_parameters

function: perturbmassesconstructorparameters
Description: Constructor for the mass perturbing merger tree operator class which takes a parameter set as input.
Code lines: 14
Contained by: file merger_trees.operators.perturb_masses.F90
Modules used: input_parameters

subroutine: perturbmassesdestructor
Description: Destructor for the mass perturbing merger tree operator function class.
Code lines: 7
Contained by: file merger_trees.operators.perturb_masses.F90

subroutine: `perturbmassesoperate`

Description: Perform a mass perturbing operation on a merger tree. Perturbations are applied to each branch of the tree, and are independent of perturbations in all other branches. Within each branch, the perturbation to each node mass is drawn from a log-normal distribution with variance and correlation specified by the selected N-body statistics class.
Code lines: 83
Contained by: file merger_trees.operators.perturb_masses.F90
Modules used: galacticus_nodes, iso_c_binding, linear_algebra, merger_tree_walkers

file: `merger_trees.operators.profiler.F90`

Description: Contains a module which implements a merger tree operator which profiles tree structure.
Code lines: 226
Modules used: cosmology_functions, kind_numbers

interface: `mergertreeoperatorprofiler`

Description: Constructors for the prune-hierarchy merger tree operator class.
Code lines: 4
Contained by: file merger_trees.operators.profiler.F90

function: `profilerconstructorinternal`

Description: Internal constructor for the information content merger tree operator class.
Code lines: 30
Contained by: file merger_trees.operators.profiler.F90
Modules used: memory_management, numerical_ranges

function: `profilerconstructorparameters`

Description: Constructor for the information content merger tree operator class which takes a parameter set as input.
Code lines: 63
Contained by: file merger_trees.operators.profiler.F90
Modules used: input_parameters

subroutine: `profilerdestructor`

Description: Destructor for the profiler merger tree operator function class.
Code lines: 7
Contained by: file merger_trees.operators.profiler.F90

subroutine: `profilerfinalize`

Description: Outputs tree information content function.
Code lines: 33
Contained by: file merger_trees.operators.profiler.F90
Modules used: galacticus_hdf5, io_hdf5

subroutine: `profileroperate`

Description: Perform a information content operation on a merger tree.
Code lines: 32
Contained by: file merger_trees.operators.profiler.F90

Modules used: galacticus_nodes merger_tree_walkers

file: merger_trees.operators.prune_baryons.F90
Description: Contains a module which implements a pruning operator on merger trees that removes all branches that can not contain any baryons.
Code lines: 158
Modules used: accretion_halos virial_density_contrast

interface: mergertreeoperatorprunebaryons
Description: Constructors for the pruning-by-mass merger tree operator class.
Code lines: 4
Contained by: file merger_trees.operators.prune_baryons.F90

function: prunebaryonsconstructorinternal
Description: Internal constructor for the prune-baryons merger tree operator class.
Code lines: 9
Contained by: file merger_trees.operators.prune_baryons.F90

function: prunebaryonsconstructorparameters
Description: Constructor for the prune-baryons merger tree operator class which takes a parameter set as input.
Code lines: 16
Contained by: file merger_trees.operators.prune_baryons.F90
Modules used: input_parameters

subroutine: prunebaryonsdestructor
Description: Destructor for the merger tree operator function class.
Code lines: 8
Contained by: file merger_trees.operators.prune_baryons.F90

subroutine: prunebaryonsoperate
Description: Prune branches from tree.
Code lines: 73
Contained by: file merger_trees.operators.prune_baryons.F90
Modules used: galacticus_nodes merger_tree_walkers merger_trees_pruning_utilities

file: merger_trees.operators.prune_branch_tips.F90
Description: Contains a module which implements a merger tree operator which prunes tips of branches (i.e. sections from the leaf node to the first node with a sibling).
Code lines: 82

interface: mergertreeoperatorprunebranchtips
Description: Constructors for the prune-branchTips merger tree operator class.
Code lines: 3
Contained by: file merger_trees.operators.prune_branch_tips.F90

function: prunebranchtipsconstructorparameters
Description: Constructor for the prune-branchTips merger tree operator class which takes a parameter set as input.

Code lines: 9
Contained by: file merger_trees.operators.prune_branch_tips.F90

subroutine: prunebranchtipsoperate
Description: Perform a prune-branchTips operation on a merger tree.
Code lines: 30
Contained by: file merger_trees.operators.prune_branch_tips.F90
Modules used: merger_tree_walkers merger_trees_pruning_utilities

file: merger_trees.operators.prune_by_mass.F90
Description: Contains a module which implements a pruning-by-mass operator on merger trees.
Code lines: 167

interface: mergertreeoperatorprunebymass
Description: Constructors for the pruning-by-mass merger tree operator class.
Code lines: 4
Contained by: file merger_trees.operators.prune_by_mass.F90

function: prunebymassconstructorinternal
Description: Internal constructor for the prune-by-mass merger tree operator class.
Code lines: 10
Contained by: file merger_trees.operators.prune_by_mass.F90

function: prunebymassconstructorparameters
Description: Constructor for the prune-by-mass merger tree operator class which takes a parameter set as input.
Code lines: 27
Contained by: file merger_trees.operators.prune_by_mass.F90
Modules used: input_parameters

subroutine: prunebymassdestructor
Description: Destructor for the merger tree operator function class.
Code lines: 8
Contained by: file merger_trees.operators.prune_by_mass.F90

subroutine: prunebymassoperate
Description: Perform a prune-by-mass operation on a merger tree.
Code lines: 73
Contained by: file merger_trees.operators.prune_by_mass.F90
Modules used: galacticus_nodes merger_tree_walkers merger_trees_pruning_utilities

file: merger_trees.operators.prune_by_time.F90
Description: Contains a module which implements a merger tree operator which prunes branches to end at a fixed time.
Code lines: 191

interface: mergertreeoperatorprunebytime
Description: Constructors for the prune-by-time merger tree operator class.

Code lines: 4
Contained by: file merger_trees.operators.prune_by_time.F90

function: prunebytimeconstructorinternal
Description: Internal constructor for the prune-by-time merger tree operator class.
Code lines: 10
Contained by: file merger_trees.operators.prune_by_time.F90

function: prunebytimeconstructorparameters
Description: Constructor for the prune-by-time merger tree operator class which takes a parameter set as input.
Code lines: 40
Contained by: file merger_trees.operators.prune_by_time.F90
Modules used: cosmology_functions

subroutine: prunebytimedestructor
Description: Destructor for the merger tree operator function class.
Code lines: 8
Contained by: file merger_trees.operators.prune_by_time.F90

subroutine: prunebytimeoperate
Description: Perform a prune-by-time operation on a merger tree.
Code lines: 85
Contained by: file merger_trees.operators.prune_by_time.F90
Modules used: galacticus_nodes merger_tree_walkers merger_trees_pruning_utilities

file: merger_trees.operators.prune_clones.F90
Description: Contains a module which implements a pruning-by-mass operator on merger trees.
Code lines: 97

interface: mergertreeoperatorpruneclones
Description: Constructors for the clone pruning merger tree operator class.
Code lines: 3
Contained by: file merger_trees.operators.prune_clones.F90

function: pruneclonesconstructorparameters
Description: Constructor for the clone pruning merger tree operator class which takes a parameter set as input.
Code lines: 10
Contained by: file merger_trees.operators.prune_clones.F90
Modules used: input_parameters

subroutine: pruneclonesdestructor
Description: Destructor for the merger tree operator function class.
Code lines: 8
Contained by: file merger_trees.operators.prune_clones.F90

subroutine: pruneclonesoperate

Description: Perform a clone pruning operation on a merger tree.
Code lines: 35
Contained by: file merger_trees.operators.prune_clones.F90
Modules used: galacticus_nodes, merger_tree_walkers, merger_trees_pruning_utilities, numerical_comparison

file: merger_trees.operators.prune_hierarchy.F90
Description: Contains a module which implements a merger tree operator which prunes branches below a given level in the substructure hierarchy.
Code lines: 155

interface: mergertreeoperatorprunehierarchy
Description: Constructors for the prune-hierarchy merger tree operator class.
Code lines: 4
Contained by: file merger_trees.operators.prune_hierarchy.F90

function: prunehierarchyconstructorinternal
Description: Internal constructor for the prune-hierarchy merger tree operator class.
Code lines: 10
Contained by: file merger_trees.operators.prune_hierarchy.F90
Modules used: galacticus_error

function: prunehierarchyconstructorparameters
Description: Constructor for the prune-hierarchy merger tree operator class which takes a parameter set as input.
Code lines: 18
Contained by: file merger_trees.operators.prune_hierarchy.F90

subroutine: prunehierarchydestructor
Description: Destructor for the merger tree operator function class.
Code lines: 8
Contained by: file merger_trees.operators.prune_hierarchy.F90

subroutine: prunehierarchyoperate
Description: Perform a prune-hierarchy operation on a merger tree.
Code lines: 60
Contained by: file merger_trees.operators.prune_hierarchy.F90
Modules used: merger_tree_walkers, merger_trees_pruning_utilities

file: merger_trees.operators.prune_lightcone.F90
Description: Contains a module which implements a prune-by-lightcone operator on merger trees.
Code lines: 205
Modules used: geometry_lightcones, satellite_oprhan_distributions

interface: mergertreeoperatorprunelightcone
Description: Constructors for the pruning-by-lightcone merger tree operator class.
Code lines: 4
Contained by: file merger_trees.operators.prune_lightcone.F90

function: `prunelightconeconstructorinternal`
Description: Internal constructor for the prune-by-lightcone merger tree operator class.
Code lines: 11
Contained by: file `merger_trees.operators.prune_lightcone.F90`

function: `prunelightconeconstructorparameters`
Description: Constructor for the prune-by-lightcone merger tree operator class which takes a parameter set as input.
Code lines: 26
Contained by: file `merger_trees.operators.prune_lightcone.F90`
Modules used: `input_parameters`

subroutine: `prunelightconedestructor`
Description: Destructor for the lightcone merger tree operator function class.
Code lines: 8
Contained by: file `merger_trees.operators.prune_lightcone.F90`

subroutine: `prunelightconeoperate`
Description: Perform a prune-by-lightcone operation on a merger tree.
Code lines: 89
Contained by: file `merger_trees.operators.prune_lightcone.F90`
Modules used: `galacticus_nodes` `histories` `merger_tree_walkers` `merger_trees_pruning_utilities`

subroutine: `prunelightconevalidate`
Description: Validate the lightcone pruning operator.
Code lines: 16
Contained by: file `merger_trees.operators.prune_lightcone.F90`
Modules used: `array_utilities` `galacticus_error` `galacticus_nodes`

file: `merger_trees.operators.prune_non_essential.F90`
Description: Contains a module which implements a merger tree operator which prunes all branches which do not contain an "essential" node.
Code lines: 158
Modules used: `kind_numbers`

interface: `mergertreeoperatorprunenonessential`
Description: Constructors for the prune-non-essential merger tree operator class.
Code lines: 4
Contained by: file `merger_trees.operators.prune_non_essential.F90`

function: `prunenonessentialconstructorinternal`
Description: Internal constructor for the prune-non-essential merger tree operator class.
Code lines: 11
Contained by: file `merger_trees.operators.prune_non_essential.F90`
Modules used: `galacticus_error`

function: `prunenonessentialconstructorparameters`
Description: Constructor for the prune-non-essential merger tree operator class which takes a parameter set as input.
Code lines: 24
Contained by: file `merger_trees.operators.prune_non_essential.F90`

subroutine: `prunenonessentialdestructor`
Description: Destructor for the merger tree operator function class.
Code lines: 8
Contained by: file `merger_trees.operators.prune_non_essential.F90`

subroutine: `prunenonessentialoperate`
Description: Perform a prune-non-essential operation on a merger tree.
Code lines: 59
Contained by: file `merger_trees.operators.prune_non_essential.F90`
Modules used: `galacticus_nodes` `merger_tree_walkers` `merger_trees_pruning_utilities`

file: `merger_trees.operators.regrid_times.F90`
Description: Contains a module which implements a merger tree operator which restructures the tree onto a fixed grid of timesteps.
Code lines: 499
Modules used: `cosmological_density_field` `cosmology_functions`

interface: `mergertreeoperatorregridtimes`
Description: Constructors for the regrid times merger tree operator class.
Code lines: 4
Contained by: file `merger_trees.operators.regrid_times.F90`

function: `regridtimesconstructorinternal`
Description: Internal constructor for the regrid times merger tree operator class.
Code lines: 59
Contained by: file `merger_trees.operators.regrid_times.F90`
Modules used: `galacticus_error` `memory_management` `numerical_ranges` `sort`

function: `regridtimesconstructorparameters`
Description: Constructor for the regrid times merger tree operator class which takes a parameter set as input.
Code lines: 89
Contained by: file `merger_trees.operators.regrid_times.F90`
Modules used: `galacticus_error`

subroutine: `regridtimesdestructor`
Description: Destructor for the merger tree operator function class.
Code lines: 8
Contained by: file `merger_trees.operators.regrid_times.F90`

subroutine: `regridtimesoperate`
Description: Perform a regrid times operation on a merger tree.
Code lines: 276
Contained by: file merger_trees.operators.regrid_times.F90
Modules used: fgsl, galacticus_error, galacticus_nodes, iso_c_binding, kind_numbers, merger_tree_walkers, merger_trees_dump, numerical_comparison, numerical_interpolation

file: `merger_trees.operators.select_within_range.F90`
Description: Contains a module which implements a select-within-range operator on the base nodes of merger trees.
Code lines: 125

interface: `mergertreeoperatorselectwithinrange`
Description: Constructors for the select-within-range merger tree operator class.
Code lines: 4
Contained by: file merger_trees.operators.select_within_range.F90

function: `selectwithinrangeconstructorinternal`
Description: Internal constructor for the select-within-range merger tree operator class.
Code lines: 9
Contained by: file merger_trees.operators.select_within_range.F90

function: `selectwithinrangeconstructorparameters`
Description: Constructor for the select-within-range merger tree operator class which takes a parameter set as input.
Code lines: 27
Contained by: file merger_trees.operators.select_within_range.F90
Modules used: input_parameters

subroutine: `selectwithinrangedestructor`
Description: Destructor for the merger tree operator function class.
Code lines: 8
Contained by: file merger_trees.operators.select_within_range.F90

subroutine: `selectwithinrangeoperate`
Description: Perform a select-within-range operation on a merger tree.
Code lines: 33
Contained by: file merger_trees.operators.select_within_range.F90
Modules used: galacticus_nodes, merger_trees_pruning_utilities

file: `merger_trees.operators.sequence.F90`
Description: Contains a module which implements a sequence of operators on merger trees.
Code lines: 168

interface: `mergertreeoperatorsequence`

Description: Constructors for the sequence merger tree operator class.
Code lines: 4
Contained by: file merger_trees.operators.sequence.F90

type: operatorlist
Code lines: 3
Contained by: file merger_trees.operators.sequence.F90

function: sequenceconstructorinternal
Description: Internal constructor for the sequence merger tree operator class.
Code lines: 14
Contained by: file merger_trees.operators.sequence.F90

function: sequenceconstructorparameters
Description: Constructor for the sequence merger tree operator class which takes a parameter set as input.
Code lines: 22
Contained by: file merger_trees.operators.sequence.F90
Modules used: input_parameters

subroutine: sequencedeepcopy
Description: Perform a deep copy for the sequence merger tree operator class.
Code lines: 31
Contained by: file merger_trees.operators.sequence.F90
Modules used: galacticus_error

subroutine: sequencedestructor
Description: Destructor for the merger tree operator function class.
Code lines: 16
Contained by: file merger_trees.operators.sequence.F90

subroutine: sequencefinalize
Description: Perform a finalization on a sequence of operators on a merger tree.
Code lines: 12
Contained by: file merger_trees.operators.sequence.F90

subroutine: sequenceoperate
Description: Perform a sequence operation on a merger tree.
Code lines: 13
Contained by: file merger_trees.operators.sequence.F90

file: merger_trees.output_structure.F90
Description: Contains a module which outputs the structure of entire merger trees.
Code lines: 287

module: merger_tree_output_structure
Description: Outputs the structure of entire merger trees.
Code lines: 265
Contained by: file merger_trees.output_structure.F90

Modules used: io_hdf5
Used by: subroutine galacticus_output_close_file subroutine evolveforestsperform

subroutine: merger_tree_structure_output
Description: Output the structure of tree.
Code lines: 228
Contained by: module merger_tree_output_structure
Modules used: dark_matter_halo_scales galacticus_hdf5 galacticus_nodes input_parameters iso_varying_string memory_management merger_tree_walkers node_component_dark_matter_profile_scale node_component_dark_matter_profile_scale_shape numerical_constants_astronomical string_handling

subroutine: merger_tree_structure_preclose
Description: Close the merger tree structure group.
Code lines: 6
Contained by: module merger_tree_output_structure

file: merger_trees.outputter.F90
Description: Contains a module which provides a class that implements evolution of merger trees.
Code lines: 58

module: merger_tree_outputters
Description: Provides a class that implements evolution of merger trees.
Code lines: 36
Contained by: file merger_trees.outputter.F90
Modules used: galacticus_nodes iso_c_binding
Used by: subroutine galacticus_function_classes_destroy subroutine galacticus_state_retrieve subroutine galacticus_state_store file tasks.evolve_forests.F90

file: merger_trees.outputter.analyzer.F90
Description: Implements a merger tree outputter class which performs analyzes on the trees.
Code lines: 140
Modules used: output_analyses

function: analyzerconstructorinternal
Description: Internal constructor for the analyzer merger tree outputter class.
Code lines: 8
Contained by: file merger_trees.outputter.analyzer.F90

function: analyzerconstructorparameters
Description: Constructor for the analyzer merger tree outputter class which takes a parameter set as input.
Code lines: 13
Contained by: file merger_trees.outputter.analyzer.F90

Modules used: input_parameters

subroutine: analyzerdestructor
Description: Destructor for the analyzer merger tree outputter class.
Code lines: 7
Contained by: file merger_trees.outputter.analyzer.F90

subroutine: analyzerfinalize
Description: Finalize merger tree output by finalizing analyses.
Code lines: 8
Contained by: file merger_trees.outputter.analyzer.F90
Modules used: galacticus_hdf5

subroutine: analyzeroutput
Description: Write properties of nodes in tree to the Galacticus output file.
Code lines: 34
Contained by: file merger_trees.outputter.analyzer.F90
Modules used: galacticus_calculations_resets galacticus_nodes merger_tree_walkers

subroutine: analyzerreduce
Description: Reduce over the outputter.
Code lines: 14
Contained by: file merger_trees.outputter.analyzer.F90
Modules used: galacticus_error

interface: mergertreeoutputteranalyzer
Description: Constructors for the analyzer merger tree outputter.
Code lines: 4
Contained by: file merger_trees.outputter.analyzer.F90

file: merger_trees.outputter.multi.F90
Description: Implements a merger trees outputter class which combines multiple other outputters.
Code lines: 194

interface: mergertreeoutputtermulti
Description: Constructors for the multi merger tree outputter.
Code lines: 4
Contained by: file merger_trees.outputter.multi.F90

function: multiconstructorinternal
Description: Internal constructor for the multi outputter class.
Code lines: 14
Contained by: file merger_trees.outputter.multi.F90

function: multiconstructorparameters
Description: Constructor for the multi merger tree outputter class which takes a parameter set as input.
Code lines: 22
Contained by: file merger_trees.outputter.multi.F90

Modules used: input_parameters

subroutine: multideepcopy
Description: Perform a deep copy for the multi outputter class.
Code lines: 31
Contained by: file merger_trees.outputter.multi.F90
Modules used: galacticus_error

subroutine: multidestructor
Description: Destructor for the multi outputter class.
Code lines: 16
Contained by: file merger_trees.outputter.multi.F90

subroutine: multifinalize
Description: Finalize all outputters.
Code lines: 12
Contained by: file merger_trees.outputter.multi.F90

subroutine: multioutput
Description: Output from all outputters.
Code lines: 16
Contained by: file merger_trees.outputter.multi.F90

type: multioutputterlist
Code lines: 3
Contained by: file merger_trees.outputter.multi.F90

subroutine: multireduce
Description: Reduce over the outputter.
Code lines: 21
Contained by: file merger_trees.outputter.multi.F90
Modules used: galacticus_error

file: merger_trees.outputter.null.F90
Description: Implements a merger trees outputter class which does no output.
Code lines: 72

interface: mergertreeoutputternull
Description: Constructors for the null merger tree outputter.
Code lines: 3
Contained by: file merger_trees.outputter.null.F90

function: nullconstructorparameters
Description: Constructor for the null merger tree outputter class which takes a parameter set as input.
Code lines: 10
Contained by: file merger_trees.outputter.null.F90
Modules used: input_parameters

subroutine: nullfinalize

Description: Finalize merger tree output.
Code lines: 7
Contained by: file merger_trees.outputter.null.F90

subroutine: nulloutput
Description: Perform no output.
Code lines: 11
Contained by: file merger_trees.outputter.null.F90

file: merger_trees.outputter.standard.F90
Description: Implements the standard class for outputting merger trees.
Code lines: 841
Modules used: cosmology_functions, galactic_filters, io_hdf5, kind_numbers, node_property_extractors

interface: mergertreeoutputterstandard
Description: Constructors for the standard merger tree outputter.
Code lines: 4
Contained by: file merger_trees.outputter.standard.F90

type: outputgroup
Description: Type used for output group information.
Code lines: 5
Contained by: file merger_trees.outputter.standard.F90

subroutine: standardbuffersallocate
Description: Allocate buffers for storage of properties.
Code lines: 36
Contained by: file merger_trees.outputter.standard.F90
Modules used: iso_c_binding, memory_management

function: standardconstructorinternal
Description: Internal constructor for the standard merger tree outputter class.
Code lines: 22
Contained by: file merger_trees.outputter.standard.F90

function: standardconstructorparameters
Description: Constructor for the standard merger tree outputter class which takes a parameter set as input.
Code lines: 37
Contained by: file merger_trees.outputter.standard.F90
Modules used: input_parameters

subroutine: standarddestructor
Description: Destructor for the standard merger tree outputter class.
Code lines: 10
Contained by: file merger_trees.outputter.standard.F90

subroutine: `standarddumpdoublebuffer`
Description: Dump the contents of the double precision properties buffer to the Galacticus output file.
Code lines: 23
Contained by: file merger_trees.outputter.standard.F90
Modules used: io_hdf5 iso_c_binding

subroutine: `standarddumpintegerbuffer`
Description: Dump the contents of the integer properties buffer to the Galacticus output file.
Code lines: 22
Contained by: file merger_trees.outputter.standard.F90
Modules used: io_hdf5

subroutine: `standardextenddoublebuffer`
Description: Extend the size of the double buffer.
Code lines: 15
Contained by: file merger_trees.outputter.standard.F90
Modules used: memory_management

subroutine: `standardextendintegerbuffer`
Description: Extend the size of the integer buffer.
Code lines: 15
Contained by: file merger_trees.outputter.standard.F90
Modules used: memory_management

subroutine: `standardfinalize`
Description: Finalize merger tree output by closing any open groups.
Code lines: 28
Contained by: file merger_trees.outputter.standard.F90
Modules used: io_hdf5

subroutine: `standardmakegroup`
Description: Make an group in the Galacticus file in which to store `tree`.
Code lines: 24
Contained by: file merger_trees.outputter.standard.F90
Modules used: galacticus_nodes iso_c_binding numerical_constants_astronomical string_handling

subroutine: `standardoutput`
Description: Write properties of nodes in `tree` to the Galacticus output file.
Code lines: 212
Contained by: file merger_trees.outputter.standard.F90
Modules used: events_hooks galacticus_calculations_resets galacticus_error galacticus_nodes input_parameters io_hdf5 iso_c_binding merger_tree_walkers multi_counters node_component_black_hole_simple node_component_black_hole_standard node_component_merging_statistics_recent

node_component_nbody_generic node_property_extractors

subroutine: standardoutputgroupcreate
Description: Create a group in which to store this output.
Code lines: 65
Contained by: file merger_trees.outputter.standard.F90
Modules used: galacticus_hdf5 io_hdf5
iso_c_binding memory_management
numerical_constants_astronomical string_handling

subroutine: standardpropertiescount
Description: Count up the number of properties that will be output.
Code lines: 69
Contained by: file merger_trees.outputter.standard.F90
Modules used: galacticus_error galacticus_nodes
node_component_black_hole_simple node_component_black_hole_standard
node_component_merging_statistics_- node_component_nbody_generic
recent
node_property_extractors

subroutine: standardpropertynamesestablish
Description: Set names for the properties.
Code lines: 88
Contained by: file merger_trees.outputter.standard.F90
Modules used: galacticus_error galacticus_nodes
node_component_black_hole_simple node_component_black_hole_standard
node_component_merging_statistics_- node_component_nbody_generic
recent
node_property_extractors

file: merger_trees.pruning.utilities.F90
Description: Contains a module which provides utility functions for pruning branches from merger trees.
Code lines: 119

module: merger_trees_pruning_utilities
Code lines: 97
Contained by: file merger_trees.pruning.utilities.F90
Used by: subroutine augmentoperate subroutine informationcontentoperate
subroutine prunebaryonsoperate subroutine prunebranchtipsoperate
subroutine prunebymassoperate subroutine prunebytimeoperate
subroutine pruneclonesoperate subroutine prunehierarchyoperate
subroutine prunelightconeoperate subroutine prunenonessentialoperate
subroutine selectwithinrangeoperate

subroutine: merger_tree_prune_clean_branch
Description: Cleans pointers in a branch about to be pruned to avoid dangling pointer problems during tree evolution.
Code lines: 27
Contained by: module merger_trees_pruning_utilities

Modules used: galacticus_nodes merger_tree_walkers

subroutine: merger_tree_prune_uniqueify_ids
Description: Ensure that nodes cloned during tree pruning have unique IDs.
Code lines: 15
Contained by: module merger_trees_pruning_utilities
Modules used: galacticus_nodes merger_tree_walkers

subroutine: merger_tree_prune_unlink_parent
Description: Unlink a parent node from a tree branch which is about to be pruned.
Code lines: 42
Contained by: module merger_trees_pruning_utilities
Modules used: galacticus_nodes

file: merger_trees.render.F90
Description: Contains a module which dumps information on merger tree structure useful for rendering 3D views of merger trees.
Code lines: 136

module: merger_trees_render
Description: Implements dumping of information on merger tree structure useful for rendering 3D views of merger trees.
Code lines: 114
Contained by: file merger_trees.render.F90
Modules used: kind_numbers

subroutine: merger_trees_render_dump
Description: Dumps information on merger tree structure useful for rendering 3D views of merger trees.
Code lines: 99
Contained by: module merger_trees_render
Modules used: cosmology_functions dark_matter_halo_scales
file_utilities galacticus_nodes
io_hdf5 memory_management
merger_tree_walkers numerical_constants_astronomical

file: merger_trees.state_store.F90
Description: Contains a module which provides state store/restore functionality for merger trees.
Code lines: 67

module: merger_tree_state_store
Description: Provides state store/restore functionality for merger trees.
Code lines: 45
Contained by: file merger_trees.state_store.F90
Modules used: iso_c_binding pseudo_random
Used by: subroutine galacticus_state_retrieve subroutine galacticus_state_store
function buildconstruct function readconstruct

subroutine: mergertreestaterestore
Description: Write the stored snapshot of the random number state to file.

Code lines: 12
Contained by: module merger_tree_state_store
Modules used: fgsl iso_c_binding

subroutine: mergertreestatestore
Description: Write the stored snapshot of the random number state to file.
Code lines: 12
Contained by: module merger_tree_state_store
Modules used: fgsl iso_c_binding

file: merger_trees.walkers.F90
Code lines: 43

module: merger_tree_walkers
Description: Provides a class of walker objects for merger trees.
Code lines: 23
Contained by: file merger_trees.walkers.F90
Modules used: galacticus_nodes
Used by:

function naozbarkana2007branchhasbaryons	function simplebranchhasbaryons
function ludlow2016formationtimeroot	function ludlow2016radius
subroutine galacticus_function_classes_destroy	subroutine meta_tree_timing_post_evolve
subroutine meta_tree_timing_pre_evolve	subroutine galacticus_state_retrieve
subroutine galacticus_state_store	subroutine cole2000build
subroutine readassignhosttreepointers	subroutine mergertreestatestore
subroutine standardevolve	subroutine merger_tree_initialize
subroutine assignorbitsoperate	function augmentaccepttree
subroutine augmentoperate	function augmenttreestatistics
subroutine conditionalmfoperate	subroutine exportoperate
subroutine informationcontentoperate	subroutine monotonizemassgrowthoperate
subroutine outputrootmassesoperate	subroutine particulateoperate
subroutine perturbmassesoperate	subroutine profileroperate
subroutine prunebaryonsoperate	subroutine prunebranchtipsoperate
subroutine prunebymassoperate	subroutine prunebytimeoperate
subroutine pruneclonesoperate	subroutine prunehierarchyoperate
subroutine prunelightconeoperate	subroutine prunenonessentialoperate
subroutine regridtimesoperate	subroutine merger_tree_structure_output
subroutine analyzeroutput	subroutine standardoutput
subroutine merger_tree_prune_clean_branch	subroutine merger_tree_prune_uniqueify_ids
subroutine merger_trees_render_dump	function mostmassiveprogenitorextract
subroutine merger_tree_dump	subroutine node_component_dark_matter_profile_scale_tree_initialize
subroutine node_component_dark_matter_profile_scale_tree_output	subroutine node_component_dark_matter_profile_scale_shape_tree_output

subroutine node_component_spin_-vitvitska_branch_initialize	function intergalacticbackgroundinternalupdate
subroutine evolveforestsperform	program tests_bug745815

file: merger_trees.walkers.all_and_formation_nodes.F90
Description: Contains a module which implements a depth-first merger tree walker over all nodes including formation nodes.
Code lines: 104

function: allandformationnodesinternal
Description: Internal constructor for the allAndFormationNodes merger tree walker class.
Code lines: 9
Contained by: file merger_trees.walkers.all_and_formation_nodes.F90

function: allandformationnodesnext
Description: Walk nodes of a tree including formation nodes.
Code lines: 26
Contained by: file merger_trees.walkers.all_and_formation_nodes.F90

function: allandformationnodesparameters
Description: Constructor for the allAndFormationNodes merger tree walker class which takes a parameter set as input.
Code lines: 11
Contained by: file merger_trees.walkers.all_and_formation_nodes.F90
Modules used: galacticus_error input_parameters

subroutine: allandformationnodesprevious
Description: Step back to the previously visited node.
Code lines: 10
Contained by: file merger_trees.walkers.all_and_formation_nodes.F90
Modules used: galacticus_error

interface: mergertreewalkerallandformationnodes
Description: Constructors for the allAndFormationNodes merger tree walker class.
Code lines: 4
Contained by: file merger_trees.walkers.all_and_formation_nodes.F90

file: merger_trees.walkers.all_nodes.F90
Description: Contains a module which implements a depth-first merger tree walker over all all nodes.
Code lines: 246
Modules used: galacticus_nodes

subroutine: allnodesdescend
Description: Descend to the deepest progenitor (satellites and children) of the current branch.
Code lines: 14
Contained by: file merger_trees.walkers.all_nodes.F90

function: allnodesinternal
Description: Internal constructor for the allNodes merger tree walker class.
Code lines: 15

Contained by: file merger_trees.walkers.all_nodes.F90

function: allnodesnext
Description: This function will update to given node to the next node which should be visited in a tree to perform a depth-first walk, including satellite nodes. Once the entire tree has been walked, a null() pointer will be set, and a value of false returned indicating that there are no more nodes to walk. Each node will be visited once and once only if the tree is walked in this way. Note that it is important that the walk descends to satellites before descending to children: the routines that destroy merger tree branches rely on this since child nodes are used in testing whether a node is a satellite—if they are destroyed prior to the test being made then problems with dangling pointers will occur.
Code lines: 88
Contained by: file merger_trees.walkers.all_nodes.F90

function: allnodesnodesremain
Description: Returns true if more nodes remain to be walked to.
Code lines: 7
Contained by: file merger_trees.walkers.all_nodes.F90

function: allnodesparameters
Description: Constructor for the allNodes merger tree walker class which takes a parameter set as input.
Code lines: 11
Contained by: file merger_trees.walkers.all_nodes.F90
Modules used: galacticus_error input_parameters

subroutine: allnodesprevious
Description: Step back to the previously visited node.
Code lines: 17
Contained by: file merger_trees.walkers.all_nodes.F90
Modules used: galacticus_error

subroutine: allnodessetnode
Description: Returns true if more nodes remain to be walked to.
Code lines: 12
Contained by: file merger_trees.walkers.all_nodes.F90

interface: mergertreewalkerallnodes
Description: Constructors for the allNodes merger tree walker class.
Code lines: 4
Contained by: file merger_trees.walkers.all_nodes.F90

file: merger_trees.walkers.all_nodes.branch.F90
Description: Contains a module which implements a depth-first merger tree walker over all all nodes in a given branch.
Code lines: 153

subroutine: allnodesbranchdescend
Description: Descend to the deepest progenitor (satellites and children) of the current branch.
Code lines: 14

Contained by: file merger_trees.walkers.all_nodes.branch.F90

function: allnodesbranchinternal
Description: Internal constructor for the allNodesBranch merger tree walker class.
Code lines: 10
Contained by: file merger_trees.walkers.all_nodes.branch.F90

function: allnodesbranchnext
Description: This function will update the given node to the next node which should be visited in a tree branch to perform a depth-first walk. Once the entire branch has been walked, a null() pointer will be set, and a value of false returned indicating that there are no more nodes to walk. Each node will be visited once and once only if the branch is walked in this way.
Code lines: 49
Contained by: file merger_trees.walkers.all_nodes.branch.F90

function: allnodesbranchnodesremain
Description: Returns true if nodes remain to be visited in the branch.
Code lines: 7
Contained by: file merger_trees.walkers.all_nodes.branch.F90

function: allnodesbranchparameters
Description: Constructor for the allNodesBranch merger tree walker class which takes a parameter set as input.
Code lines: 11
Contained by: file merger_trees.walkers.all_nodes.branch.F90
Modules used: galacticus_error input_parameters

interface: mergertreewalkerallnodesbranch
Description: Constructors for the allNodesBranch merger tree walker class.
Code lines: 4
Contained by: file merger_trees.walkers.all_nodes.branch.F90

file: merger_trees.walkers.isolated_nodes.F90
Description: Contains a module which implements a depth-first merger tree walker over all isolated nodes.
Code lines: 134

function: isolatednodesinternal
Description: Internal constructor for the isolatedNodes merger tree walker class.
Code lines: 9
Contained by: file merger_trees.walkers.isolated_nodes.F90

function: isolatednodesnext
Description: This function will update to given node to the next node which should be visited in a tree to perform a depth-first walk. Once the entire tree has been walked, a null() pointer will be set, and a value of false returned indicating that there are no more nodes to walk. Each node will be visited once and once only if the tree is walked in this way.
Code lines: 70
Contained by: file merger_trees.walkers.isolated_nodes.F90

function: isolatednodesparameters

Description: Constructor for the `isolatedNodes` merger tree walker class which takes a parameter set as input.
Code lines: 11
Contained by: file `merger_trees.walkers.isolated_nodes.F90`
Modules used: `galacticus_error` `input_parameters`

interface: `mergertreewalkerisolatednodes`
Description: Constructors for the `isolatedNodes` merger tree walker class.
Code lines: 4
Contained by: file `merger_trees.walkers.isolated_nodes.F90`

file: `merger_trees.walkers.isolated_nodes.branch.F90`
Description: Contains a module which implements a depth-first merger tree walker over all isolated nodes in a given branch.
Code lines: 126

function: `isolatednodesbranchinternal`
Description: Internal constructor for the `isolatedNodesBranch` merger tree walker class.
Code lines: 13
Contained by: file `merger_trees.walkers.isolated_nodes.branch.F90`

function: `isolatednodesbranchnext`
Description: This function will update the given `node` to the next node which should be visited in a tree branch to perform a depth-first walk. Once the entire branch has been walked, a `null()` pointer will be set, and a value of `false` returned indicating that there are no more nodes to walk. Each node will be visited once and once only if the branch is walked in this way.
Code lines: 45
Contained by: file `merger_trees.walkers.isolated_nodes.branch.F90`
Modules used: `galacticus_nodes`

function: `isolatednodesbranchnodesremain`
Description: Returns true if nodes remain to be visited in the branch.
Code lines: 7
Contained by: file `merger_trees.walkers.isolated_nodes.branch.F90`

function: `isolatednodesbranchparameters`
Description: Constructor for the `isolatedNodesBranch` merger tree walker class which takes a parameter set as input.
Code lines: 11
Contained by: file `merger_trees.walkers.isolated_nodes.branch.F90`
Modules used: `galacticus_error` `input_parameters`

interface: `mergertreewalkerisolatednodesbranch`
Description: Constructors for the `isolatedNodesBranch` merger tree walker class.
Code lines: 4
Contained by: file `merger_trees.walkers.isolated_nodes.branch.F90`

file: `merger_trees.walkers.tree_construction.F90`
Description: Contains a module which implements a tree walker for trees under construction.
Code lines: 126

Modules used: galacticus_nodes

interface: mergertreewalkertreeconstruction
Description: Constructors for the treeConstruction merger tree walker class.
Code lines: 4
Contained by: file merger_trees.walkers.tree_construction.F90

function: treeconstructioninternal
Description: Internal constructor for the treeConstruction merger tree walker class.
Code lines: 10
Contained by: file merger_trees.walkers.tree_construction.F90

function: treeconstructionnext
Code lines: 47
Contained by: file merger_trees.walkers.tree_construction.F90

function: treeconstructionnodesremain
Description: Returns true if more nodes remain to be walked to.
Code lines: 7
Contained by: file merger_trees.walkers.tree_construction.F90

function: treeconstructionparameters
Description: Constructor for the treeConstruction merger tree walker class which takes a parameter set as input.
Code lines: 11
Contained by: file merger_trees.walkers.tree_construction.F90
Modules used: galacticus_error input_parameters

file: meta.tree_processing_times.F90
Description: Contains a module providing a class for calculations of the time taken to process merger trees.
Code lines: 40

module: meta_tree_compute_times
Description: Provides a class for calculations of the time taken to process merger trees.
Code lines: 18
Contained by: file meta.tree_processing_times.F90
Used by: subroutine galacticus_function_classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; file merger_trees.construct.build.masses.distribution.stellar_mass_function.F90

file: meta.tree_processing_times.file.F90
Description: Contains a module which implements a merger tree processing time estimator using a polynomial relation read from file.
Code lines: 97

function: fileconstructorinternal
Description: Internal constructor for the “file” merger tree processing time estimator class.

Code lines: 19
Contained by: file meta.tree_processing_times.file.F90
Modules used: fox_dom galacticus_error io_xml

function: fileconstructorparameters
Description: Constructor for the "file" merger tree processing time estimator class which takes a parameter set as input.
Code lines: 17
Contained by: file meta.tree_processing_times.file.F90
Modules used: input_parameters

function: filetime
Description: Return the units of the file property in the SI system.
Code lines: 15
Contained by: file meta.tree_processing_times.file.F90

interface: metatreeprocessingtimefile
Description: Constructors for the "file" merger tree processing time estimator.
Code lines: 4
Contained by: file meta.tree_processing_times.file.F90

file: models.likelihoods.F90
Description: Contains a module which implements a likelihood class for posterior sampling simulations.
Code lines: 75

module: models_likelihoods
Description: Implements a likelihood class for posterior sampling simulations.
Code lines: 53
Contained by: file models.likelihoods.F90
Modules used: model_parameters posterior_sampling_convergence posterior_sampling_state
Used by: subroutine galacticus_function_classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; file posterior_sampling.simulation.differential_evolution.F90; file posterior_sampling.simulation.particle_swarm.F90; module posterior_sampling_state_initialize

file: models.likelihoods.SED_fit.F90
Description: Implementation of a posterior sampling likelihood class which implements a likelihood for SED fitting.
Code lines: 471
Modules used: cosmology_functions stellar_population_selectors stellar_population_spectra_postprocess

interface: posteriorsamplelikelihoodsedfit

Description: Constructors for the `sedFit` posterior sampling convergence class.
Code lines: 4
Contained by: file models.likelihoods.SED_fit.F90

function: `sedfitconstructorinternal`
Description: Constructor for "sedFit" posterior sampling likelihood class.
Code lines: 44
Contained by: file models.likelihoods.SED_fit.F90
Modules used: galacticus_error, instruments_filters, iso_varying_string, memory_management

function: `sedfitconstructorparameters`
Description: Constructor for the `sedFit` posterior sampling convergence class which builds the object from a parameter set.
Code lines: 72
Contained by: file models.likelihoods.SED_fit.F90
Modules used: input_parameters

subroutine: `sedfitdestructor`
Description: Destructor for "sedFit" posterior sampling likelihood class.
Code lines: 9
Contained by: file models.likelihoods.SED_fit.F90

function: `sedfitevaluate`
Description: Return the log-likelihood for the SED fitting likelihood function.
Code lines: 251
Contained by: file models.likelihoods.SED_fit.F90
Modules used: abundances_structure, cosmology_functions, fgsl, galactic_structure_options, galacticus_error, galacticus_nodes, iso_c_binding, models_likelihoods_constants, numerical_integration, posterior_sampling_convergence, posterior_sampling_state, rapidevaluation, stellar_population_luminosities, stellar_populations, stellar_spectra_dust_attenuations

function: `luminosityintegrand`
Description: Star formation rate integrand.
Code lines: 31
Contained by: function sedfitevaluate
Modules used: numerical_constants_math, stellar_population_luminosities

subroutine: `sedfitfunctionchanged`
Description: Respond to possible changes in the likelihood function.
Code lines: 7
Contained by: file models.likelihoods.SED_fit.F90

file: `models.likelihoods.constants.F90`
Description: Contains a module which provides constants for use when constraining Galacticus.

Code lines: 30

module: `models_likelihoods_constants`
Description: Provides constants for use when constraining Galacticus.
Code lines: 8
Contained by: file models.likelihoods.constants.F90
Used by:

function	function sedfitevaluate
morphologicalfractiongamamoffett2016loglikelihood	
function galaxypopulationevaluate	function galaxypopulationwillevaluate
function gaussianregressionevaluate	subroutine gaussianregressionrestore
function gaussianregressionwillevaluate	function halomassfunctionevaluate
function	function massfunctionevaluate
independentlikelihoodssequantialevaluate	
function	function spindistributionevaluate
projectedcorrelationfunctionevaluate	
function modelparameterlistlogprior	function activelogprior
subroutine	subroutine particleswarmposterior
differentialevolutionsimulate	
subroutine particleswarmsimulate	subroutine latinhypercubeinitialize
subroutine priorrandominitialize	subroutine resumeinitialize
subroutine switchedinitialize	

file: `models.likelihoods.galaxy_population.F90`
Description: Implementation of a posterior sampling likelihood class which implements a likelihood for Galacticus models.
Code lines: 427
Modules used:

fox_dom	input_parameters
output_analyses	

function: `galaxypopulationconstructorinternal`
Description: Constructor for "galaxyPopulation" posterior sampling likelihood class.
Code lines: 11
Contained by: file models.likelihoods.galaxy_population.F90

function: `galaxypopulationconstructorparameters`
Description: Constructor for the `galaxyPopulation` posterior sampling likelihood class which builds the object from a parameter set.
Code lines: 58
Contained by: file models.likelihoods.galaxy_population.F90
Modules used:

galacticus_display	input_parameters

subroutine: `galaxypopulationdestructor`
Description: Destructor for the `galaxyPopulation` posterior sampling likelihood class.
Code lines: 8
Contained by: file models.likelihoods.galaxy_population.F90

function: `galaxypopulationevaluate`
Description: Return the log-likelihood for the Galacticus likelihood function.
Code lines: 255

Contained by: file models.likelihoods.galaxy_population.F90
Modules used: functions_global, galacticus_display, galacticus_error, kind_numbers, models_likelihoods_constants, mpi_utilities, posterior_sampling_convergence, posterior_sampling_state, string_handling

subroutine: galaxypopulationfunctionchanged
Description: Respond to possible changes in the likelihood function.
Code lines: 7
Contained by: file models.likelihoods.galaxy_population.F90

function: galaxypopulationwillevaluate
Description: Return true if the log-likelihood will be evaluated.
Code lines: 16
Contained by: file models.likelihoods.galaxy_population.F90
Modules used: models_likelihoods_constants, posterior_sampling_convergence, posterior_sampling_state

type: inputparameterlist
Description: Type used to maintain a list of pointers to parameters to be modified.
Code lines: 6
Contained by: file models.likelihoods.galaxy_population.F90

interface: posteriorsamplelikelihoodgalaxypopulation
Description: Constructors for the galaxyPopulation posterior sampling likelihood class.
Code lines: 4
Contained by: file models.likelihoods.galaxy_population.F90

file: models.likelihoods.gaussian_regression.F90
Description: Implementation of a posterior sampling likelihood class which implements a likelihood using Gaussian regression to emulate another likelihood.
Code lines: 1029
Modules used: fgsl

function: gaussianregressionconstructorinternal
Description: Constructor for "gaussianRegression" posterior sampling likelihood class.
Code lines: 24
Contained by: file models.likelihoods.gaussian_regression.F90
Modules used: file_utilities, galacticus_error

function: gaussianregressionconstructorparameters
Description: Constructor for the gaussianRegression posterior sampling likelihood class which builds the object from a parameter set.
Code lines: 90
Contained by: file models.likelihoods.gaussian_regression.F90
Modules used: input_parameters

function: gaussianregressioncorrelation

Description: Compute correlation of the variogram model.
Code lines: 8
Contained by: file models.likelihoods.gaussian_regression.F90

subroutine: gaussianregressiondestructor
Description: Destructor for Gaussian regression likelihood class.
Code lines: 9
Contained by: file models.likelihoods.gaussian_regression.F90

subroutine: gaussianregressionemulate
Description: Evaluate the model emulator.
Code lines: 48
Contained by: file models.likelihoods.gaussian_regression.F90
Modules used: fgsl

function: gaussianregressionevaluate
Description: Return the log-likelihood for a Gaussian regression likelihood function.
Code lines: 322
Contained by: file models.likelihoods.gaussian_regression.F90
Modules used:

dates_and_times	error_functions
galacticus_display	galacticus_error
iso_c_binding	memory_management
models_likelihoods_constants	mpi_utilities
posterior_sampling_convergence	posterior_sampling_state
string_handling	

function: gaussianregressionevaluatevariogram
Description: Compute the variogram at separation h.
Code lines: 14
Contained by: file models.likelihoods.gaussian_regression.F90

subroutine: gaussianregressionfitvariogram
Description: Compute best fit coefficients for the variogram model.
Code lines: 96
Contained by: file models.likelihoods.gaussian_regression.F90
Modules used:

fgsl	galacticus_error
iso_c_binding	sort

subroutine: gaussianregressionfunctionchanged
Description: Respond to possible changes in the likelihood function.
Code lines: 28
Contained by: file models.likelihoods.gaussian_regression.F90
Modules used: memory_management

subroutine: gaussianregressionrestore
Description: Process a previous state to restore likelihood function.
Code lines: 31
Contained by: file models.likelihoods.gaussian_regression.F90

Modules used: memory_management models_likelihoods_constants

function: gaussianregressionseparation
Description: Determine the separation between two state vectors.
Code lines: 8
Contained by: file models.likelihoods.gaussian_regression.F90

subroutine: gaussianregressionvariogrammodeld
Description: Derivatives of the function to be minimized when fitting the variogram.
Code lines: 27
Contained by: file models.likelihoods.gaussian_regression.F90
Modules used: fgsl iso_c_binding

function: gaussianregressionvariogrammodelf
Description: Function to be minimized when fitting the variogram.
Code lines: 21
Contained by: file models.likelihoods.gaussian_regression.F90
Modules used: fgsl iso_c_binding

subroutine: gaussianregressionvariogrammodelfd
Description: Computes both function and derivatives to be minimized when fitting the variogram.
Code lines: 10
Contained by: file models.likelihoods.gaussian_regression.F90
Modules used: iso_c_binding

function: gaussianregressionwillevaluate
Description: Return true if the log-likelihood will be evaluated.
Code lines: 47
Contained by: file models.likelihoods.gaussian_regression.F90
Modules used: galacticus_display iso_c_binding
models_likelihoods_constants posterior_sampling_convergence
posterior_sampling_state

function: polynomialcoefficientcount
Description: Return the number of coefficients at $\mathtt{n}^{\mathrm{th}}$ order in polynomial of dimension d.
Code lines: 12
Contained by: file models.likelihoods.gaussian_regression.F90
Modules used: factorials

interface: polynomialiterator
Code lines: 2
Contained by: file models.likelihoods.gaussian_regression.F90

function: polynomialiteratorconstructor
Description: Create a polynomial iterator for a poynomial of specified order and rank.
Code lines: 12
Contained by: file models.likelihoods.gaussian_regression.F90
Modules used: memory_management

function: `polynomialiteratorcounter`
Description: Return the current count of a polynomial iterator.
Code lines: 7
Contained by: file `models.likelihoods.gaussian_regression.F90`

function: `polynomialiteratorcurrentorder`
Description: Return the current order of a polynomial iterator.
Code lines: 7
Contained by: file `models.likelihoods.gaussian_regression.F90`

function: `polynomialiteratorindex`
Description: Return the requested index of a polynomial iterator.
Code lines: 8
Contained by: file `models.likelihoods.gaussian_regression.F90`

function: `polynomialiteratoriterate`
Description: Iterate over polynomial coefficients.
Code lines: 31
Contained by: file `models.likelihoods.gaussian_regression.F90`

subroutine: `polynomialiteratorreset`
Description: Reset a polynomial iterator.
Code lines: 10
Contained by: file `models.likelihoods.gaussian_regression.F90`

interface: `posteriorsamplelikelihoodgaussianregression`
Description: Constructors for the `gaussianRegression` posterior sampling likelihood class.
Code lines: 4
Contained by: file `models.likelihoods.gaussian_regression.F90`

file: `models.likelihoods.halo_mass_function.F90`
Description: Implementation of a posterior sampling likelihood class which implements a likelihood for halo mass functions.
Code lines: 428
Modules used: `cosmological_density_field` `cosmology_functions` `cosmology_parameters` `dark_matter_halo_scales` `dark_matter_profiles_dmo`

function: `halomassfunctionconstructorinternal`
Description: Constructor for "haloMassFunction" posterior sampling likelihood class.
Code lines: 104
Contained by: file `models.likelihoods.halo_mass_function.F90`
Modules used: `galacticus_display` `galacticus_error` `io_hdf5` `memory_management`

function: `halomassfunctionconstructorparameters`
Description: Constructor for the `haloMassFunction` posterior sampling convergence class which builds the object from a parameter set.
Code lines: 98

Contained by: file models.likelihoods.halo_mass_function.F90
Modules used: input_parameters

subroutine: halomassfunctiondestructor
Description: Destructor for "haloMassFunction" posterior sampling likelihood class.
Code lines: 15
Contained by: file models.likelihoods.halo_mass_function.F90

function: halomassfunctionevaluate
Description: Return the log-likelihood for the halo mass function likelihood function.
Code lines: 120
Contained by: file models.likelihoods.halo_mass_function.F90
Modules used: galacticus_error, halo_mass_functions, models_likelihoods_constants, mpi_utilities, posterior_sampling_convergence, posterior_sampling_state, statistics_nbody_halo_mass_errors

subroutine: halomassfunctionfunctionchanged
Description: Respond to possible changes in the likelihood function.
Code lines: 7
Contained by: file models.likelihoods.halo_mass_function.F90

interface: posteriorsamplelikelihoodhalomassfunction
Description: Constructors for the haloMassFunction posterior sampling convergence class.
Code lines: 4
Contained by: file models.likelihoods.halo_mass_function.F90

file: models.likelihoods.independent_likelihoods.F90
Description: Implementation of a model likelihood class which combines other likelihoods assumed to be independent.
Code lines: 227

function: independentlikelihoodsconstructorinternal
Description: Constructor for "independentLikelihoods" posterior sampling likelihood class.
Code lines: 8
Contained by: file models.likelihoods.independent_likelihoods.F90

function: independentlikelihoodsconstructorparameters
Description: Constructor for the independentLikelihoods posterior sampling convergence class which builds the object from a parameter set.
Code lines: 49
Contained by: file models.likelihoods.independent_likelihoods.F90
Modules used: galacticus_error, input_parameters, string_handling

subroutine: independentlikelihoodsdestructor
Description: Destructor for "independentLikelihoods" posterior sampling likelihood class.
Code lines: 22
Contained by: file models.likelihoods.independent_likelihoods.F90

function: `independentlikelihoodsevaluate`
Description: Return the log-likelihood for the halo mass function likelihood function.
Code lines: 75
Contained by: file `models.likelihoods.independent_likelihoods.F90`
Modules used: `galacticus_error`

subroutine: `independentlikelihoodsfunctionchanged`
Description: Respond to possible changes in the likelihood function.
Code lines: 12
Contained by: file `models.likelihoods.independent_likelihoods.F90`

interface: `posteriorsamplelikelihoodindependentlikelihoods`
Description: Constructors for the `independentLikelihoods` posterior sampling convergence class.
Code lines: 4
Contained by: file `models.likelihoods.independent_likelihoods.F90`

type: `posteriorsamplelikelihoodlist`
Code lines: 8
Contained by: file `models.likelihoods.independent_likelihoods.F90`

file: `models.likelihoods.independent_likelihoods.sequential.F90`
Description: Implementation of a model likelihood class which combines other likelihoods assumed to be independent.
Code lines: 306

function: `independentlikelihoodssequantialevaluate`
Description: Return the log-likelihood for the halo mass function likelihood function.
Code lines: 167
Contained by: file `models.likelihoods.independent_likelihoods.sequential.F90`
Modules used: `galacticus_display`, `galacticus_error`, `iso_varying_string`, `models_likelihoods_constants`, `mpi_utilities`, `string_handling`

function: `independentlikelihoodssequentialconstructorinternal`
Description: Constructor for "independentLikelihoods" posterior sampling likelihood class.
Code lines: 15
Contained by: file `models.likelihoods.independent_likelihoods.sequential.F90`

function: `independentlikelihoodssequentialconstructorparameters`
Description: Constructor for the `independentLikelihoods` posterior sampling convergence class which builds the object from a parameter set.
Code lines: 44
Contained by: file `models.likelihoods.independent_likelihoods.sequential.F90`
Modules used: `galacticus_error`, `input_parameters`

subroutine: `independentlikelihoodssequentialrestore`
Description: Process a previous state to restore progress state.
Code lines: 17

Contained by: file models.likelihoods.independent_likelihoods.sequential.F90

interface: posteriorsamplelikelihoodindpndntlklhdssqntl
Description: Constructors for the independentLikelihoods posterior sampling convergence class.
Code lines: 4
Contained by: file models.likelihoods.independent_likelihoods.sequential.F90

file: models.likelihoods.mass_function.F90
Description: Implementation of a posterior sampling likelihood class which implements a likelihood for mass functions.
Code lines: 366
Modules used: cosmology_functions, geometry_surveys, halo_mass_functions

function: massfunctionconstructorinternal
Description: Constructor for "massFunction" posterior sampling likelihood class.
Code lines: 67
Contained by: file models.likelihoods.mass_function.F90
Modules used: galacticus_display, galacticus_paths, io_hdf5, memory_management

function: massfunctionconstructorparameters
Description: Constructor for the massFunction posterior sampling convergence class which builds the object from a parameter set.
Code lines: 79
Contained by: file models.likelihoods.mass_function.F90
Modules used: input_parameters

subroutine: massfunctiondestructor
Description: Destructor for "massFunction" posterior sampling likelihood class.
Code lines: 9
Contained by: file models.likelihoods.mass_function.F90

function: massfunctionevaluate
Description: Return the log-likelihood for the mass function likelihood function.
Code lines: 140
Contained by: file models.likelihoods.mass_function.F90
Modules used: conditional_mass_functions, fgsl, galacticus_error, mass_function_incompletenesses, models_likelihoods_constants, numerical_integration, posterior_sampling_convergence, posterior_sampling_state

function: likelihoodmassfunctionhalomassintegrand
Description: Integral over halo mass function.
Code lines: 9
Contained by: function massfunctionevaluate

function: likelihoodmassfunctiontimeintegrand
Description: Integral over time.
Code lines: 38

Contained by: function massfunctionevaluate
Modules used: galacticus_error string_handling

function: likelihoodmassfunctiontimenormalizationintegrand
Description: Normalization integral over time.
Code lines: 7
Contained by: function massfunctionevaluate

subroutine: massfunctionfunctionchanged
Description: Respond to possible changes in the likelihood function.
Code lines: 7
Contained by: file models.likelihoods.mass_function.F90

interface: posteriorsamplelikelihoodmassfunction
Description: Constructors for the massFunction posterior sampling convergence class.
Code lines: 4
Contained by: file models.likelihoods.mass_function.F90

file: models.likelihoods.multivariate_normal.F90
Description: Implementation of a posterior sampling likelihood class which implements a multivariate normal likelihood.
Code lines: 127
Modules used: linear_algebra

function: multivariatenormalconstructorinternal
Description: Constructor for "multivariateNormal" convergence class.
Code lines: 10
Contained by: file models.likelihoods.multivariate_normal.F90

function: multivariatenormalconstructorparameters
Description: Constructor for the multivariateNormal posterior sampling convergence class which builds the object from a parameter set.
Code lines: 29
Contained by: file models.likelihoods.multivariate_normal.F90
Modules used: input_parameters

function: multivariatenormalevaluate
Description: Return the log-likelihood for a multivariate-normal likelihood function.
Code lines: 30
Contained by: file models.likelihoods.multivariate_normal.F90
Modules used: posterior_sampling_convergence posterior_sampling_state

subroutine: multivariatenormalfunctionchanged
Description: Respond to possible changes in the likelihood function.
Code lines: 7
Contained by: file models.likelihoods.multivariate_normal.F90

interface: posteriorsamplelikelihoodmultivariatenormal
Description: Constructors for the multivariateNormal posterior sampling convergence class.
Code lines: 4

Contained by: file models.likelihoods.multivariate_normal.F90

file: models.likelihoods.multivariate_normal.stochastic.F90
Description: Implementation of a posterior sampling likelihood class which implements a multivariate normal likelihood.
Code lines: 179
Modules used: pseudo_random

function: multivariatenormalstochasticconstructorinternal
Description: Constructor for "multivariateNormalStochastic" convergence class.
Code lines: 10
Contained by: file models.likelihoods.multivariate_normal.stochastic.F90

function: multivariatenormalstochasticconstructorparameters
Description: Constructor for the multivariateNormalStochastic posterior sampling convergence class which builds the object from a parameter set.
Code lines: 45
Contained by: file models.likelihoods.multivariate_normal.stochastic.F90
Modules used: input_parameters iso_varying_string

function: multivariatenormalstochasticevaluate
Description: Return the log-likelihood for a multivariate-normal likelihood function.
Code lines: 66
Contained by: file models.likelihoods.multivariate_normal.stochastic.F90
Modules used: numerical_constants_math posterior_sampling_convergence posterior_sampling_state

subroutine: multivariatenormalstochasticfunctionchanged
Description: Respond to possible changes in the likelihood function.
Code lines: 7
Contained by: file models.likelihoods.multivariate_normal.stochastic.F90

interface: posteriorsamplelikelihoodmltivrtnormalstochastic
Description: Constructors for the multivariateNormalStochastic posterior sampling convergence class.
Code lines: 4
Contained by: file models.likelihoods.multivariate_normal.stochastic.F90

file: models.likelihoods.posterior_as_prior.F90
Description: Implementation of a posterior sampling likelihood class which implements a likelihood using a given posterior distribution over the parameters in the form of a set of MCMC chains.
Code lines: 301
Modules used: nearest_neighbors

function: posterioraspriorconstructorinternal
Description: Constructor for "posteriorAsPrior" posterior sampling likelihood class.
Code lines: 87
Contained by: file models.likelihoods.posterior_as_prior.F90
Modules used: file_utilities iso_varying_string memory_management string_handling

function: posterioraspriorconstructorparameters
Description: Constructor for the posteriorAsPrior posterior sampling likelihood class which builds the object from a parameter set.
Code lines: 47
Contained by: file models.likelihoods.posterior_as_prior.F90
Modules used: input_parameters

subroutine: posterioraspriordestructor
Description: Destructor for "posterior as prior" posterior sampling likelihood class.
Code lines: 7
Contained by: file models.likelihoods.posterior_as_prior.F90

function: posterioraspriorevaluate
Description: Return the log-likelihood for a "posterior as prior" likelihood function.
Code lines: 35
Contained by: file models.likelihoods.posterior_as_prior.F90
Modules used: numerical_constants_math posterior_sampling_convergence posterior_sampling_state

subroutine: posteriorasprioriorfunctionchanged
Description: Respond to possible changes in the likelihood function.
Code lines: 7
Contained by: file models.likelihoods.posterior_as_prior.F90

subroutine: posterioraspriorinitialize
Description: Initialize a "posterior as prior" likelihood object.
Code lines: 43
Contained by: file models.likelihoods.posterior_as_prior.F90
Modules used: memory_management

interface: posteriorsamplelikelihoodposterioraspriorior
Description: Constructors for the posteriorAsPrior posterior sampling likelihood class.
Code lines: 4
Contained by: file models.likelihoods.posterior_as_prior.F90

file: models.likelihoods.projected_correlation_function.F90
Description: Implementation of a posterior sampling likelihood class which implements a likelihood for projected correlation functions.
Code lines: 269
Modules used: cosmology_functions dark_matter_halo_biases dark_matter_halo_scales dark_matter_profile_scales dark_matter_profiles_dmo geometry_surveys halo_mass_functions linear_algebra linear_growth power_spectra

interface: posteriorsamplelikelihoodprjctdcorrelationfunction
Description: Constructors for the projectedCorrelationFunction posterior sampling convergence class.
Code lines: 4

Contained by: file models.likelihoods.projected_correlation_function.F90

function: projectedcorrelationfunctionconstructorinternal
Description: Constructor for "projectedCorrelationFunction" posterior sampling likelihood class.
Code lines: 46
Contained by: file models.likelihoods.projected_correlation_function.F90
Modules used: galacticus_paths, io_hdf5, memory_management, node_component_dark_matter_profile_scale

function: projectedcorrelationfunctionconstructorparameters
Description: Constructor for the projectedCorrelationFunction posterior sampling convergence class which builds the object from a parameter set.
Code lines: 76
Contained by: file models.likelihoods.projected_correlation_function.F90
Modules used: input_parameters

subroutine: projectedcorrelationfunctiondestructor
Description: Destructor for the projectedCorrelationFunction class.
Code lines: 15
Contained by: file models.likelihoods.projected_correlation_function.F90

function: projectedcorrelationfunctionevaluate
Description: Return the log-likelihood for the projected correlation function likelihood function.
Code lines: 48
Contained by: file models.likelihoods.projected_correlation_function.F90
Modules used: conditional_mass_functions, galacticus_error, halo_model_projected_correlations, models_likelihoods_constants, posterior_sampling_convergence, posterior_sampling_state

subroutine: projectedcorrelationfunctionfunctionchanged
Description: Respond to possible changes in the likelihood function.
Code lines: 7
Contained by: file models.likelihoods.projected_correlation_function.F90

file: models.likelihoods.spin_distribution.F90
Description: Implementation of a posterior sampling likelihood class which implements a likelihood for halo spin distributions.
Code lines: 314
Modules used: cosmology_functions, dark_matter_halo_scales, dark_matter_profile_scales, dark_matter_profiles_dmo, halo_mass_functions, statistics_nbody_halo_mass_errors

interface: posteriorsamplelikelihoodspindistribution
Description: Constructors for the spinDistribution posterior sampling convergence class.
Code lines: 4
Contained by: file models.likelihoods.spin_distribution.F90

function: spindistributionconstructorinternal

Description: Constructor for "spinDistribution" posterior sampling likelihood class.
Code lines: 41
Contained by: file models.likelihoods.spin_distribution.F90
Modules used: cosmology_functions, galacticus_error, io_hdf5, memory_management

function: spindistributionconstructorparameters
Description: Constructor for the spinDistribution posterior sampling convergence class which builds the object from a parameter set.
Code lines: 81
Contained by: file models.likelihoods.spin_distribution.F90
Modules used: input_parameters

subroutine: spindistributiondestructor
Description: Destructor for "spinDistribution" posterior sampling likelihood class.
Code lines: 12
Contained by: file models.likelihoods.spin_distribution.F90

function: spindistributionevaluate
Description: Return the log-likelihood for the halo spin distribution likelihood function.
Code lines: 98
Contained by: file models.likelihoods.spin_distribution.F90
Modules used: fgsl, galacticus_error, galacticus_nodes, halo_spin_distributions, iso_c_binding, models_likelihoods_constants, mpi_utilities, numerical_integration, posterior_sampling_convergence, posterior_sampling_state

function: spindistributionintegrate
Description: Integrand function used to find cumulative spin distribution over a bin.
Code lines: 8
Contained by: function spindistributionevaluate

subroutine: spindistributionfunctionchanged
Description: Respond to possible changes in the likelihood function.
Code lines: 7
Contained by: file models.likelihoods.spin_distribution.F90

file: models.parameters.F90
Description: Contains a module that implements a class of parameter mapping functions.
Code lines: 115

module: model_parameters
Description: Implements a class of unary operators.
Code lines: 93
Contained by: file models.parameters.F90
Used by: subroutine galacticus_function_classes_destroy, subroutine galacticus_state_retrieve

subroutine galacticus_state_store
module posterior_sample_differential_-random_jump
file posterior_-sampling.simulation.particle_-swarm.F90
module models_likelihoods
file posterior_-sampling.simulation.differential_-evolution.F90
module posterior_sampling_state_-initialize

type: modelparameterlist
Description: Class used to construct lists of model parameters.
Code lines: 3
Contained by: module model_parameters

function: modelparameterlistlogprior
Description: Compute the log-prior of a list of parameters.
Code lines: 23
Contained by: module model_parameters
Modules used: models_likelihoods_constants posterior_sampling_state

file: models.parameters.active.F90
Description: Implementation of an active model parameter class.
Code lines: 198
Modules used: math_operators_unary statistics_distributions

function: activeconstructorinternal
Description: Internal constructor for the active model parameter class.
Code lines: 10
Contained by: file models.parameters.active.F90

function: activeconstructorparameters
Description: Constructor for the active model parameter class which builds the object from a parameter set.
Code lines: 27
Contained by: file models.parameters.active.F90
Modules used: input_parameters

subroutine: activedestructor
Description: Destructor for "active" model parameter class.
Code lines: 9
Contained by: file models.parameters.active.F90

function: activelogprior
Description: Return the log-prior on this parameter.
Code lines: 14
Contained by: file models.parameters.active.F90
Modules used: models_likelihoods_constants

function: activemap
Description: Map this parameter.

Code lines: 8
Contained by: file models.parameters.active.F90

function: activename
Description: Return the name of this parameter.
Code lines: 8
Contained by: file models.parameters.active.F90

function: activepriorinvert
Description: Invert the prior, returning the parameter value given the cumulative probability.
Code lines: 8
Contained by: file models.parameters.active.F90

function: activepriormaximum
Description: Return the maximum value for which the prior is non-zero.
Code lines: 7
Contained by: file models.parameters.active.F90

function: activepriorminimum
Description: Return the minimum value for which the prior is non-zero.
Code lines: 7
Contained by: file models.parameters.active.F90

function: activepriorsample
Description: Sample from the of this parameter.
Code lines: 7
Contained by: file models.parameters.active.F90

function: activerandomperturbation
Description: Return a random perturbation to this parameter.
Code lines: 7
Contained by: file models.parameters.active.F90

function: activeunmap
Description: Unmap this parameter.
Code lines: 8
Contained by: file models.parameters.active.F90

interface: modelparameteractive
Description: Constructors for the active 1D distribution function class.
Code lines: 4
Contained by: file models.parameters.active.F90

file: models.parameters.derived.F90
Description: Implementation of a model parameter class in which the parameter value is derived from other parameters.
Code lines: 96

function: derivedconstructorinternal

Description: Internal constructor for the `derived` model parameter class.
Code lines: 8
Contained by: file `models.parameters.derived.F90`

function: `derivedconstructorparameters`
Description: Constructor for the `derived` model parameter class which builds the object from a parameter set.
Code lines: 25
Contained by: file `models.parameters.derived.F90`
Modules used: `input_parameters`

function: `deriveddefinition`
Description: Return the definition of this parameter.
Code lines: 8
Contained by: file `models.parameters.derived.F90`

interface: `modelparameterderived`
Description: Constructors for the `derived` 1D distribution function class.
Code lines: 4
Contained by: file `models.parameters.derived.F90`

file: `models.parameters.inactive.F90`
Description: Implementation of an inactive model parameter class.
Code lines: 189

function: `inactiveconstructorinternal`
Description: Internal constructor for the `inactive` model parameter class.
Code lines: 8
Contained by: file `models.parameters.inactive.F90`

function: `inactiveconstructorparameters`
Description: Constructor for the `inactive` model parameter class which builds the object from a parameter set.
Code lines: 19
Contained by: file `models.parameters.inactive.F90`
Modules used: `input_parameters`

function: `inactivelogprior`
Description: Return the log-prior on this parameter.
Code lines: 11
Contained by: file `models.parameters.inactive.F90`
Modules used: `galacticus_error`

function: `inactivemap`
Description: Map this parameter.
Code lines: 11
Contained by: file `models.parameters.inactive.F90`
Modules used: `galacticus_error`

function: `inactivename`
Description: Return the name of this parameter.
Code lines: 8
Contained by: file `models.parameters.inactive.F90`

function: `inactivepriorinvert`
Description: Invert the prior, returning the parameter value given the cumulative probability.
Code lines: 11
Contained by: file `models.parameters.inactive.F90`
Modules used: `galacticus_error`

function: `inactivepriormaximum`
Description: Return the maximum value for which the prior is non-zero.
Code lines: 10
Contained by: file `models.parameters.inactive.F90`
Modules used: `galacticus_error`

function: `inactivepriorminimum`
Description: Return the minimum value for which the prior is non-zero.
Code lines: 10
Contained by: file `models.parameters.inactive.F90`
Modules used: `galacticus_error`

function: `inactivepriorsample`
Description: Sample from the of this parameter.
Code lines: 10
Contained by: file `models.parameters.inactive.F90`
Modules used: `galacticus_error`

function: `inactiverandomperturbation`
Description: Return a random perturbation to this parameter.
Code lines: 10
Contained by: file `models.parameters.inactive.F90`
Modules used: `galacticus_error`

function: `inactiveunmap`
Description: Unmap this parameter.
Code lines: 11
Contained by: file `models.parameters.inactive.F90`
Modules used: `galacticus_error`

interface: `modelparameterinactive`
Description: Constructors for the `inactive` 1D distribution function class.
Code lines: 4
Contained by: file `models.parameters.inactive.F90`

file: `nBody.import.F90`
Description: Contains a module which provides a class that implements importing of data from N-body simulations.

Code lines: 41

module: `nbody_importers`
Description: Provides a class that implements importing of data from N-body simulations.
Code lines: 19
Contained by: file `nBody.import.F90`
Modules used: `nbody_simulation_data`
Used by: subroutine `galacticus_function_-classes_destroy`; subroutine `galacticus_state_retrieve`; subroutine `galacticus_state_store`; file `tasks.nBody_analyze.F90`

file: `nBody.import.GadgetBinary.F90`
Description: Contains a module which implements an N-body data importer for Gadget binary files.
Code lines: 179
Modules used: `io_hdf5`; `iso_varying_string`

function: `gadgetbinaryconstructorinternal`
Description: Internal constructor for the "gadgetBinary" N-body importer class.
Code lines: 10
Contained by: file `nBody.import.GadgetBinary.F90`

function: `gadgetbinaryconstructorparameters`
Description: Constructor for the "gadgetBinary" N-body importer class which takes a parameter set as input.
Code lines: 62
Contained by: file `nBody.import.GadgetBinary.F90`
Modules used: `input_parameters`

function: `gadgetbinaryimport`
Description: Import data from a Gadget HDF5 file.
Code lines: 57
Contained by: file `nBody.import.GadgetBinary.F90`
Modules used: `galacticus_error`; `memory_management`; `numerical_constants_astronomical`; `numerical_constants_prefixes`

interface: `nbodyimportergadgetbinary`
Description: Constructors for the "gadgetBinary" N-body importer class.
Code lines: 4
Contained by: file `nBody.import.GadgetBinary.F90`

file: `nBody.import.GadgetHDF5.F90`
Description: Contains a module which implements an N-body data importer for Gadget HDF5 files.
Code lines: 171
Modules used: `io_hdf5`

function: `gadgethdf5constructorinternal`
Description: Internal constructor for the "gadgetHDF5" N-body importer class.
Code lines: 9
Contained by: file `nBody.import.GadgetHDF5.F90`

function: `gadgethdf5constructorparameters`
Description: Constructor for the "gadgetHDF5" N-body importer class which takes a parameter set as input.
Code lines: 47
Contained by: file `nBody.import.GadgetHDF5.F90`
Modules used: `input_parameters`

subroutine: `gadgethdf5destructor`
Description: Destructor for Gadget HF5 importer class.
Code lines: 7
Contained by: file `nBody.import.GadgetHDF5.F90`

function: `gadgethdf5import`
Description: Import data from a Gadget HDF5 file.
Code lines: 56
Contained by: file `nBody.import.GadgetHDF5.F90`
Modules used: `galacticus_error` `numerical_constants_astronomical` `numerical_constants_prefixes`

interface: `nbodyimportergadgethdf5`
Description: Constructors for the "gadgetHDF5" N-body importer class.
Code lines: 4
Contained by: file `nBody.import.GadgetHDF5.F90`

file: `nBody.operator.F90`
Description: Contains a module which provides a class that implements operators on data from N-body simulations.
Code lines: 40

module: `nbody_operators`
Description: Provides a class that implements operators on data from N-body simulations.
Code lines: 18
Contained by: file `nBody.operator.F90`
Modules used: `nbody_simulation_data`
Used by: subroutine `galacticus_function_-classes_destroy`; subroutine `galacticus_state_retrieve`; subroutine `galacticus_state_store`; file `tasks.nBody_analyze.F90`

file: `nBody.operator.environmental_overdensity.F90`
Description: Contains a module which implements an N-body data operator which determines the environmental overoverdensity around particles.
Code lines: 235
Modules used: `iso_c_binding`

function: `environmentaloverdensityconstructorinternal`
Description: Internal constructor for the "environmentalOverdensity" N-body operator class.
Code lines: 13
Contained by: file `nBody.operator.environmental_overdensity.F90`

Modules used: numerical_constants_math

function: environmentaloverdensityconstructorparameters
Description: Constructor for the "environmentalOverdensity" N-body operator class which takes a parameter set as input.
Code lines: 51
Contained by: file nBody.operator.environmental_overdensity.F90
Modules used: input_parameters

subroutine: environmentaloverdensityoperate
Description: Determine the mean position and velocity of N-body particles.
Code lines: 121
Contained by: file nBody.operator.environmental_overdensity.F90
Modules used: galacticus_display memory_management nearest_neighbors omp_lib

function: boundlower
Description: Compute lower bounds for particle inclusion in periodically replicated volumes.
Code lines: 18
Contained by: subroutine environmentaloverdensityoperate
Modules used: galacticus_error

function: boundupper
Description: Compute upper bounds for particle inclusion in periodically replicated volumes.
Code lines: 18
Contained by: subroutine environmentaloverdensityoperate
Modules used: galacticus_error

interface: nbodyoperatorenvironmentaloverdensity
Description: Constructors for the "environmentalOverdensity" N-body operator class.
Code lines: 4
Contained by: file nBody.operator.environmental_overdensity.F90

file: nBody.operator.mean_position.F90
Description: Contains a module which implements an N-body data operator which determines the mean position and velocity of particles.
Code lines: 137
Modules used: iso_c_binding

function: meanpositionconstructorinternal
Description: Internal constructor for the "meanPosition" N-body operator class.
Code lines: 10
Contained by: file nBody.operator.mean_position.F90
Modules used: input_parameters

function: meanpositionconstructorparameters
Description: Constructor for the "meanPosition" N-body operator class which takes a parameter set as input.
Code lines: 27
Contained by: file nBody.operator.mean_position.F90

Modules used: input_parameters

subroutine: meanpositionoperate
Description: Determine the mean position and velocity of N-body particles.
Code lines: 51
Contained by: file nBody.operator.mean_position.F90
Modules used: galacticus_error memory_management pseudo_random

interface: nbodyoperatormeanposition
Description: Constructors for the "meanPosition" N-body operator class.
Code lines: 4
Contained by: file nBody.operator.mean_position.F90

file: nBody.operator.null.F90
Description: Contains a module which implements a null N-body data operator.
Code lines: 60

interface: nbodyoperatornull
Description: Constructors for the "null" N-body operator class.
Code lines: 3
Contained by: file nBody.operator.null.F90

function: nullconstructorparameters
Description: Constructor for the "null" N-body operator class which takes a parameter set as input.
Code lines: 10
Contained by: file nBody.operator.null.F90
Modules used: input_parameters

subroutine: nulloperate
Description: Perform a null operation on N-body simulation data.
Code lines: 9
Contained by: file nBody.operator.null.F90

file: nBody.operator.pair_counts.F90
Description: Contains a module which implements an N-body data operator which computes pair counts in bins of separation.
Code lines: 172
Modules used: iso_c_binding

interface: nbodyoperatorpaircounts
Description: Constructors for the "pairCounts" N-body operator class.
Code lines: 4
Contained by: file nBody.operator.pair_counts.F90

function: paircountsconstructorinternal
Description: Internal constructor for the "pairCounts" N-body operator class.
Code lines: 9
Contained by: file nBody.operator.pair_counts.F90

function: `paircountsconstructorparameters`
Description: Constructor for the "pairCounts" N-body operator class which takes a parameter set as input.
Code lines: 49
Contained by: file `nBody.operator.pair_counts.F90`
Modules used: `input_parameters`

subroutine: `paircountsoperate`
Description: Determine the mean position and velocity of N-body particles.
Code lines: 65
Contained by: file `nBody.operator.pair_counts.F90`
Modules used: `galacticus_display` `memory_management` `nearest_neighbors` `numerical_ranges` `omp_lib` `pseudo_random`

file: `nBody.operator.rotation_curve.F90`
Description: Contains a module which implements an N-body data operator which computes the rotation curve at a set of given radii.
Code lines: 166
Modules used: `iso_c_binding`

interface: `nbodyoperatorrotationcurve`
Description: Constructors for the "rotationCurve" N-body operator class.
Code lines: 4
Contained by: file `nBody.operator.rotation_curve.F90`

function: `rotationcurveconstructorinternal`
Description: Internal constructor for the "rotationCurve" N-body operator class.
Code lines: 10
Contained by: file `nBody.operator.rotation_curve.F90`

function: `rotationcurveconstructorparameters`
Description: Constructor for the "rotationCurve" N-body operator class which takes a parameter set as input.
Code lines: 36
Contained by: file `nBody.operator.rotation_curve.F90`
Modules used: `input_parameters`

subroutine: `rotationcurveoperate`
Description: Determine the mean position and velocity of N-body particles.
Code lines: 70
Contained by: file `nBody.operator.rotation_curve.F90`
Modules used: `galacticus_error` `io_hdf5` `memory_management` `numerical_constants_physical` `pseudo_random`

file: `nBody.operator.self_bound.F90`
Description: Contains a module which implements an N-body data operator which determines the subset of particles that are self-bound.

Code lines: 492
Modules used: `iso_c_binding`

interface: `nbodyoperatorselfbound`
Description: Constructors for the "selfBound" N-body operator class.
Code lines: 4
Contained by: file `nBody.operator.self_bound.F90`

function: `selfboundconstructorinternal`
Description: Internal constructor for the "selfBound" N-body operator class
Code lines: 11
Contained by: file `nBody.operator.self_bound.F90`
Modules used: `input_parameters`

function: `selfboundconstructorparameters`
Description: Constructor for the "selfBound" N-body operator class which takes a parameter set as input.
Code lines: 52
Contained by: file `nBody.operator.self_bound.F90`
Modules used: `input_parameters`

subroutine: `selfboundoperate`
Description: Determine the subset of N-body particles which are self-bound.
Code lines: 360
Contained by: file `nBody.operator.self_bound.F90`
Modules used: `galacticus_display` `galacticus_error` `iso_varying_string` `memory_management` `numerical_constants_physical` `pseudo_random` `string_handling`

function: `selfboundpotential`
Description: Compute the potential for an array of particle separations. Currently assumes the functional form of the softening used by Gadget.
Code lines: 17
Contained by: file `nBody.operator.self_bound.F90`

file: `nBody.operator.sequence.F90`
Description: Contains a module which implements an N-body data operator which applies a sequence of other operators.
Code lines: 153

type: `nbodyoperatorlist`
Code lines: 3
Contained by: file `nBody.operator.sequence.F90`

interface: `nbodyoperatorsequence`
Description: Constructors for the "sequence" N-body operator class.
Code lines: 4
Contained by: file `nBody.operator.sequence.F90`

function: `sequenceconstructorinternal`
Description: Internal constructor for the "sequence" N-body operator class.
Code lines: 15
Contained by: file `nBody.operator.sequence.F90`
Modules used: `input_parameters`

function: `sequenceconstructorparameters`
Description: Constructor for the "sequence" N-body operator class which takes a parameter set as input.
Code lines: 22
Contained by: file `nBody.operator.sequence.F90`
Modules used: `input_parameters`

subroutine: `sequencedeepcopy`
Description: Perform a deep copy for the `sequence` N-body operator class.
Code lines: 31
Contained by: file `nBody.operator.sequence.F90`
Modules used: `galacticus_error`

subroutine: `sequencedestructor`
Description: Destructor for the sequence N-body operator class.
Code lines: 16
Contained by: file `nBody.operator.sequence.F90`

subroutine: `sequenceoperate`
Description: Apply a sequence of N-body simulation operators.
Code lines: 13
Contained by: file `nBody.operator.sequence.F90`

file: `nBody.operator.velocity_dispersion.F90`
Description: Contains a module which implements an N-body data operator which computes the velocity dispersion in a set of given spherical shells.
Code lines: 188
Modules used: `iso_c_binding`

interface: `nbodyoperatorvelocitydispersion`
Description: Constructors for the "velocityDispersion" N-body operator class.
Code lines: 4
Contained by: file `nBody.operator.velocity_dispersion.F90`

function: `velocitydispersionconstructorinternal`
Description: Internal constructor for the "velocityDispersion" N-body operator class.
Code lines: 12
Contained by: file `nBody.operator.velocity_dispersion.F90`
Modules used: `galacticus_error`

function: `velocitydispersionconstructorparameters`
Description: Constructor for the "velocityDispersion" N-body operator class which takes a parameter set as input.
Code lines: 44

Contained by: file nBody.operator.velocity_dispersion.F90
Modules used: input_parameters

subroutine: velocitydispersionoperate
Description: Determine the mean position and velocity of N-body particles.
Code lines: 82
Contained by: file nBody.operator.velocity_dispersion.F90
Modules used: galacticus_error, io_hdf5, memory_management, numerical_constants_physical, pseudo_random

file: nodes.property_extractor.F90
Description: Contains a module which provides a class that implements on-the-fly analyses.
Code lines: 61

module: node_property_extractors
Description: Provides a class that implements extraction of properties from nodes.
Code lines: 39
Contained by: file nodes.property_extractor.F90
Modules used: galacticus_nodes, multi_counters, output_analyses_options
Used by: subroutine galacticus_function_classes_destroy; function hivshalomassrelationpadmanabhan2017constructorin; function localgroupmassfunctionconstructorinternal; function blackholebulgerelationconstructorinternal; function concentrationvshalomasscdmludlow2016constructorinternal; file galacticus.output.analyses.correlation_function.F90; file galacticus.output.analyses.distribution_operator.random_error.N-body_concentration.F90; function massmetallicityandrews2013constructorinternal; function massmetallicityblanc2017constructorinternal; file galacticus.output.analyses.mean_function_1d.F90; function morphologicalfractiongamamoffett2016constructorinternal; file galacticus.output.analyses.scatter_function_1d.F90; function stellarvshalomassrelationleauthaud2012constructorinternal; file galacticus.output.analyses.volume_function_1d.F90; file galacticus.output.analyses.weight_operator.normal.F90; file galacticus.output.analyses.weight_operator.property.F90; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; file merger_trees.outputter.standard.F90; subroutine standardoutput

subroutine standardpropertiescount

subroutine standardpropertynamesestablish

file: nodes.property_extractor.ICM_SZ.F90

Description: Contains a module which implements an intracluster medium Sunyaev-Zeldovich property extractor class.
Code lines: 211
Modules used: chemical_states, cosmology_functions, dark_matter_halo_scales, hot_halo_mass_distributions, hot_halo_temperature_profiles

function: icmszconstructorinternal

Description: Internal constructor for the icmSZ property extractor class.
Code lines: 12
Contained by: file nodes.property_extractor.ICM_SZ.F90

function: icmszconstructorparameters

Description: Constructor for the icmSZ property extractor class which takes a parameter set as input.
Code lines: 25
Contained by: file nodes.property_extractor.ICM_SZ.F90
Modules used: input_parameters

function: icmszdescription

Description: Return a description of the intracluster medium Sunyaev-Zeldovich property.
Code lines: 9
Contained by: file nodes.property_extractor.ICM_SZ.F90

subroutine: icmszdestructor

Description: Destructor for the icmSZ property extractor class.
Code lines: 11
Contained by: file nodes.property_extractor.ICM_SZ.F90

function: icmszextract

Description: Implement a Sunyaev-Zeldovich effect property extractor.
Code lines: 57
Contained by: file nodes.property_extractor.ICM_SZ.F90
Modules used: fgsl, galacticus_nodes, numerical_constants_math, numerical_integration, radiation_fields

function: integrandcomptiony

Description: Integrand function used for computing ICM SZ properties.
Code lines: 27
Contained by: function icmszextract
Modules used: abundances_structure, numerical_constants_astronomical, numerical_constants_atomic, numerical_constants_physical, numerical_constants_prefixes

function: icmszname

Description: Return the name of the last isolated redshift property.
Code lines: 9
Contained by: file nodes.property_extractor.ICM_SZ.F90

function: icmsztype
Description: Return the type of the last isolated redshift property.
Code lines: 9
Contained by: file nodes.property_extractor.ICM_SZ.F90
Modules used: output_analyses_options

function: icmszunitsinsi
Description: Return the units of the last isolated redshift property in the SI system.
Code lines: 8
Contained by: file nodes.property_extractor.ICM_SZ.F90

interface: nodepropertyextractoricmsz
Description: Constructors for the "icmSZ" output analysis class.
Code lines: 4
Contained by: file nodes.property_extractor.ICM_SZ.F90

file: nodes.property_extractor.ICM_Xray_luminosity.F90
Description: Contains a module which implements an intracluster medium X-ray luminosity property extractor class.
Code lines: 271
Modules used: cooling_functions, cosmology_functions, dark_matter_halo_scales, hot_halo_mass_distributions, hot_halo_temperature_profiles

function: icmxrayluminosityconstructorinternal
Description: Internal constructor for the icmXRayLuminosity property extractor class.
Code lines: 12
Contained by: file nodes.property_extractor.ICM_Xray_luminosity.F90

function: icmxrayluminosityconstructorparameters
Description: Constructor for the icmXRayLuminosity property extractor class which takes a parameter set as input.
Code lines: 25
Contained by: file nodes.property_extractor.ICM_Xray_luminosity.F90
Modules used: input_parameters

function: icmxrayluminositydescriptions
Description: Return descriptions of the icmXRayLuminosity properties.
Code lines: 11
Contained by: file nodes.property_extractor.ICM_Xray_luminosity.F90

subroutine: icmxrayluminositydestructor
Description: Destructor for the icmXRayLuminosity property extractor class.
Code lines: 11
Contained by: file nodes.property_extractor.ICM_Xray_luminosity.F90

function: `icmxrayluminosityelementcount`
Description: Return the number of elements in the lightconeple property extractors.
Code lines: 9
Contained by: file nodes.property_extractor.ICM_Xray_luminosity.F90

function: `icmxrayluminosityextract`
Description: Implement an ICM X-ray properties extractor.
Code lines: 96
Contained by: file nodes.property_extractor.ICM_Xray_luminosity.F90
Modules used: `fgsl` galacticus_nodes numerical_constants_physical numerical_constants_prefixes numerical_constants_units numerical_integration radiation_fields

function: `integrandluminosityxray`
Description: Integrand function used for computing ICM X-ray luminosities.
Code lines: 40
Contained by: function icmxrayluminosityextract
Modules used: abundances_structure chemical_abundances_structure chemical_reaction_rates_utilities numerical_constants_astronomical numerical_constants_atomic numerical_constants_math numerical_constants_prefixes

function: `integrandtemperaturexray`
Description: Integrand function used for computing ICM X-ray luminosity-weighted temperatures.
Code lines: 7
Contained by: function icmxrayluminosityextract

function: `icmxrayluminositynames`
Description: Return the names of the `icmXRayLuminosity` properties.
Code lines: 11
Contained by: file nodes.property_extractor.ICM_Xray_luminosity.F90

function: `icmxrayluminositytype`
Description: Return the type of the `icmXRayLuminosity` properties.
Code lines: 9
Contained by: file nodes.property_extractor.ICM_Xray_luminosity.F90
Modules used: output_analyses_options

function: `icmxrayluminosityunitsinsi`
Description: Return the units of the `icmXRayLuminosity` properties in the SI system.
Code lines: 13
Contained by: file nodes.property_extractor.ICM_Xray_luminosity.F90
Modules used: numerical_constants_prefixes numerical_constants_units

interface: `nodepropertyextractoricmxrayluminosity`
Description: Constructors for the "icmXRayLuminosity" output analysis class.
Code lines: 4

Contained by: file nodes.property_extractor.ICM_Xray_luminosity.F90

file: nodes.property_extractor.cold_mode_infall_rate.F90
Description: Contains a module which implements a cold mode infall rate property extractor class.
Code lines: 138
Modules used: cooling_cold_mode_infall_rates

interface: nodepropertyextractorrateinfallcoldmode
Description: Constructors for the rateInfallColdMode output analysis class.
Code lines: 4
Contained by: file nodes.property_extractor.cold_mode_infall_rate.F90

function: rateinfallcoldmodeconstructorinternal
Description: Internal constructor for the rateInfallColdMode property extractor class.
Code lines: 8
Contained by: file nodes.property_extractor.cold_mode_infall_rate.F90

function: rateinfallcoldmodeconstructorparameters
Description: Constructor for the rateInfallColdMode property extractor class which takes a parameter set as input.
Code lines: 13
Contained by: file nodes.property_extractor.cold_mode_infall_rate.F90
Modules used: input_parameters

function: rateinfallcoldmodedescription
Description: Return a description of the rateInfallColdMode property.
Code lines: 9
Contained by: file nodes.property_extractor.cold_mode_infall_rate.F90

subroutine: rateinfallcoldmodedestructor
Description: Destructor for the rateInfallColdMode property extractor class.
Code lines: 7
Contained by: file nodes.property_extractor.cold_mode_infall_rate.F90

function: rateinfallcoldmodeextract
Description: Implement a rateInfallColdMode property extractor.
Code lines: 11
Contained by: file nodes.property_extractor.cold_mode_infall_rate.F90
Modules used: galacticus_nodes

function: rateinfallcoldmodename
Description: Return the name of the rateInfallColdMode property.
Code lines: 9
Contained by: file nodes.property_extractor.cold_mode_infall_rate.F90

function: rateinfallcoldmodetype
Description: Return the type of the rateInfallColdMode property.
Code lines: 9
Contained by: file nodes.property_extractor.cold_mode_infall_rate.F90

Modules used: output_analyses_options

function: rateinfallcoldmodeunitsinsi
Description: Return the units of the rateInfallColdMode property in the SI system.
Code lines: 9
Contained by: file nodes.property_extractor.cold_mode_infall_rate.F90
Modules used: numerical_constants_astronomical

file: nodes.property_extractor.concentration.F90
Description: Contains a module which implements a concentration output analysis property extractor class.
Code lines: 148
Modules used: virial_density_contrast

function: concentrationconstructorinternal
Description: Internal constructor for the "concentration" output analysis property extractor class.
Code lines: 9
Contained by: file nodes.property_extractor.concentration.F90
Modules used: input_parameters

function: concentrationconstructorparameters
Description: Constructor for the "concentration" output analysis property extractor class which takes a parameter set as input.
Code lines: 13
Contained by: file nodes.property_extractor.concentration.F90
Modules used: input_parameters

function: concentrationdescription
Description: Return a description of the concentration property.
Code lines: 9
Contained by: file nodes.property_extractor.concentration.F90

subroutine: concentrationdestructor
Description: Destructor for the concentration output analysis property extractor class.
Code lines: 7
Contained by: file nodes.property_extractor.concentration.F90

function: concentrationextract
Description: Implement a concentration output analysis.
Code lines: 18
Contained by: file nodes.property_extractor.concentration.F90
Modules used: dark_matter_profile_mass_definitions galacticus_nodes

function: concentrationname
Description: Return the name of the concentration property.
Code lines: 9
Contained by: file nodes.property_extractor.concentration.F90

function: concentrationtype
Description: Return the type of the concentration property.

Code lines: 9
Contained by: file nodes.property_extractor.concentration.F90
Modules used: output_analyses_options

function: concentrationunitsinsi
Description: Return the units of the concentration property in the SI system.
Code lines: 8
Contained by: file nodes.property_extractor.concentration.F90

interface: nodepropertyextractorconcentration
Description: Constructors for the "concentration" output analysis class.
Code lines: 4
Contained by: file nodes.property_extractor.concentration.F90

file: nodes.property_extractor.cooling_radius.F90
Description: Contains a module which implements a radiusCooling property extractor class.
Code lines: 137
Modules used: cooling_radii

interface: nodepropertyextractorradiuscooling
Description: Constructors for the "radiusCooling" output analysis class.
Code lines: 4
Contained by: file nodes.property_extractor.cooling_radius.F90

function: radiuscoolingconstructorinternal
Description: Internal constructor for the radiusCooling property extractor class.
Code lines: 8
Contained by: file nodes.property_extractor.cooling_radius.F90

function: radiuscoolingconstructorparameters
Description: Constructor for the radiusCooling property extractor class which takes a parameter set as input.
Code lines: 13
Contained by: file nodes.property_extractor.cooling_radius.F90
Modules used: input_parameters

function: radiuscoolingdescription
Description: Return a description of the cooling radius property.
Code lines: 9
Contained by: file nodes.property_extractor.cooling_radius.F90

subroutine: radiuscoolingdestructor
Description: Destructor for the radiusCooling property extractor class.
Code lines: 7
Contained by: file nodes.property_extractor.cooling_radius.F90

function: radiuscoolingextract
Description: Implement a cooling radius property extractor.
Code lines: 10

Contained by: file nodes.property_extractor.cooling_radius.F90

function: radiuscoolingname
Description: Return the name of the cooling radius property.
Code lines: 9
Contained by: file nodes.property_extractor.cooling_radius.F90

function: radiuscoolingtype
Description: Return the type of the last isolated redshift property.
Code lines: 9
Contained by: file nodes.property_extractor.cooling_radius.F90
Modules used: output_analyses_options

function: radiuscoolingunitsinsi
Description: Return the units of the cooling radius property in the SI system.
Code lines: 9
Contained by: file nodes.property_extractor.cooling_radius.F90
Modules used: numerical_constants_astronomical

file: nodes.property_extractor.cooling_rate.F90
Description: Contains a module which implements a cooling rate property extractor class.
Code lines: 138
Modules used: cooling_rates

interface: nodepropertyextractorratecooling
Description: Constructors for the "rateCooling" output analysis class.
Code lines: 4
Contained by: file nodes.property_extractor.cooling_rate.F90

function: ratecoolingconstructorinternal
Description: Internal constructor for the rateCooling property extractor class.
Code lines: 8
Contained by: file nodes.property_extractor.cooling_rate.F90

function: ratecoolingconstructorparameters
Description: Constructor for the rateCooling property extractor class which takes a parameter set as input.
Code lines: 13
Contained by: file nodes.property_extractor.cooling_rate.F90
Modules used: input_parameters

function: ratecoolingdescription
Description: Return a description of the rateCooling property.
Code lines: 9
Contained by: file nodes.property_extractor.cooling_rate.F90

subroutine: ratecoolingdestructor
Description: Destructor for the rateCooling property extractor class.
Code lines: 7

Contained by: file nodes.property_extractor.cooling_rate.F90

function: ratecoolingextract
Description: Implement a last isolated redshift output analysis.
Code lines: 11
Contained by: file nodes.property_extractor.cooling_rate.F90
Modules used: galacticus_nodes

function: ratecoolingname
Description: Return the name of the last isolated redshift property.
Code lines: 9
Contained by: file nodes.property_extractor.cooling_rate.F90

function: ratecoolingtype
Description: Return the type of the last isolated redshift property.
Code lines: 9
Contained by: file nodes.property_extractor.cooling_rate.F90
Modules used: output_analyses_options

function: ratecoolingunitsinsi
Description: Return the units of the last isolated redshift property in the SI system.
Code lines: 9
Contained by: file nodes.property_extractor.cooling_rate.F90
Modules used: numerical_constants_astronomical

file: nodes.property_extractor.density.F90
Description: Contains a module which implements a property extractor class for the density at a set of radii.
Code lines: 241
Modules used: dark_matter_halo_scales galactic_structure_radii_definitions

function: densityprofileconstructorinternal
Description: Internal constructor for the densityProfile property extractor class.
Code lines: 19
Contained by: file nodes.property_extractor.density.F90
Modules used: galactic_structure_radii_definitions

function: densityprofileconstructorparameters
Description: Constructor for the densityProfile property extractor class which takes a parameter set as input.
Code lines: 31
Contained by: file nodes.property_extractor.density.F90
Modules used: input_parameters

function: densityprofiledescriptions
Description: Return descriptions of the densityProfile property.
Code lines: 15
Contained by: file nodes.property_extractor.density.F90

subroutine: densityprofiledestructor
Description: Destructor for the densityProfile property extractor class.
Code lines: 7
Contained by: file nodes.property_extractor.density.F90

function: densityprofileelementcount
Description: Return the number of elements in the densityProfile property extractors.
Code lines: 9
Contained by: file nodes.property_extractor.density.F90

function: densityprofileextract
Description: Implement a densityProfile property extractor.
Code lines: 50
Contained by: file nodes.property_extractor.density.F90
Modules used: galactic_structure_densities, galactic_structure_enclosed_masses, galactic_structure_options, galactic_structure_radii_definitions, galacticus_nodes

function: densityprofilenames
Description: Return the names of the densityProfile properties.
Code lines: 15
Contained by: file nodes.property_extractor.density.F90

function: densityprofiletype
Description: Return the type of the densityProfile properties.
Code lines: 9
Contained by: file nodes.property_extractor.density.F90
Modules used: output_analyses_options

function: densityprofileunitsinsi
Description: Return the units of the densityProfile properties in the SI system.
Code lines: 16
Contained by: file nodes.property_extractor.density.F90
Modules used: numerical_constants_astronomical

interface: nodepropertyextractordensityprofile
Description: Constructors for the "densityProfile" output analysis class.
Code lines: 4
Contained by: file nodes.property_extractor.density.F90

file: nodes.property_extractor.density_contrasts.F90
Description: Contains a module which implements a property extractor class for the mass and radii of spheres are specified density contrast.
Code lines: 269
Modules used: cosmology_functions, cosmology_parameters, dark_matter_halo_scales, galacticus_nodes

function: densitycontrastsconstructorinternal

Description: Internal constructor for the `densityContrasts` property extractor class.
Code lines: 23
Contained by: file nodes.property_extractor.density_contrasts.F90
Modules used: galactic_structure_options

function: `densitycontrastsconstructorparameters`
Description: Constructor for the `densityContrasts` property extractor class which takes a parameter set as input.
Code lines: 37
Contained by: file nodes.property_extractor.density_contrasts.F90
Modules used: input_parameters

function: `densitycontrastsdescriptions`
Description: Return descriptions of the `densityContrasts` property.
Code lines: 18
Contained by: file nodes.property_extractor.density_contrasts.F90

subroutine: `densitycontrastsdestructor`
Description: Destructor for the `densityContrasts` property extractor class.
Code lines: 9
Contained by: file nodes.property_extractor.density_contrasts.F90

function: `densitycontrastselementcount`
Description: Return the number of elements in the `densityContrasts` property extractors.
Code lines: 9
Contained by: file nodes.property_extractor.density_contrasts.F90

function: `densitycontrastsextract`
Description: Implement a last isolated redshift output analysis.
Code lines: 36
Contained by: file nodes.property_extractor.density_contrasts.F90
Modules used: galactic_structure_enclosed_masses galactic_structure_options root_finder

function: `densitycontrastsnames`
Description: Return the names of the `densityContrasts` properties.
Code lines: 18
Contained by: file nodes.property_extractor.density_contrasts.F90

function: `densitycontrastsroot`
Description: Root function used in finding the radius that encloses a given density contrast.
Code lines: 12
Contained by: file nodes.property_extractor.density_contrasts.F90
Modules used: galactic_structure_enclosed_masses galactic_structure_options numerical_constants_math

function: `densitycontraststype`
Description: Return the type of the `densityContrasts` properties.
Code lines: 9

Contained by: file nodes.property_extractor.density_contrasts.F90
Modules used: output_analyses_options

function: densitycontrastsunitsinsi
Description: Return the units of the densityContrasts properties in the SI system.
Code lines: 15
Contained by: file nodes.property_extractor.density_contrasts.F90
Modules used: numerical_constants_astronomical

interface: nodepropertyextractordensitycontrasts
Description: Constructors for the "densityContrasts" output analysis class.
Code lines: 4
Contained by: file nodes.property_extractor.density_contrasts.F90

file: nodes.property_extractor.descendents.F90
Description: Contains a module which implements an ISM mass output analysis property extractor class.
Code lines: 190
Modules used: output_times

function: descendentsconstructorinternal
Description: Internal constructor for the descendents node property extractor class.
Code lines: 8
Contained by: file nodes.property_extractor.descendents.F90

function: descendentsconstructorparameters
Description: Constructor for the descendents node property extractor class which takes a parameter set as input.
Code lines: 13
Contained by: file nodes.property_extractor.descendents.F90
Modules used: input_parameters

function: descendentsdescription
Description: Return a description of the descendents property.
Code lines: 9
Contained by: file nodes.property_extractor.descendents.F90

subroutine: descendentsdestructor
Description: Destructor for the descendents property extractor class.
Code lines: 7
Contained by: file nodes.property_extractor.descendents.F90

function: descendentsextract
Description: Implement a descendents node property extractor.
Code lines: 76
Contained by: file nodes.property_extractor.descendents.F90
Modules used: galacticus_nodes

function: descendentsname
Description: Return the name of the descendents property.

Code lines: 9
Contained by: file nodes.property_extractor.descendents.F90

function: descendentstype
Description: Return the type of the stellar mass property.
Code lines: 9
Contained by: file nodes.property_extractor.descendents.F90
Modules used: output_analyses_options

interface: nodepropertyextractordescendents
Description: Constructors for the "descendents" output analysis class.
Code lines: 4
Contained by: file nodes.property_extractor.descendents.F90

file: nodes.property_extractor.final_descendent.F90
Description: Contains a module which implements an ISM mass output analysis property extractor class.
Code lines: 111

function: finaldescendentconstructorparameters
Description: Constructor for the finalDescendent node property extractor class which takes a parameter set as input.
Code lines: 10
Contained by: file nodes.property_extractor.final_descendent.F90
Modules used: input_parameters

function: finaldescendentdescription
Description: Return a description of the finalDescendent property.
Code lines: 9
Contained by: file nodes.property_extractor.final_descendent.F90

function: finaldescendentextract
Description: Implement a finalDescendent node property extractor.
Code lines: 21
Contained by: file nodes.property_extractor.final_descendent.F90

function: finaldescendentname
Description: Return the name of the finalDescendent property.
Code lines: 9
Contained by: file nodes.property_extractor.final_descendent.F90

function: finaldescendenttype
Description: Return the type of the stellar mass property.
Code lines: 9
Contained by: file nodes.property_extractor.final_descendent.F90
Modules used: output_analyses_options

interface: nodepropertyextractorfinaldescendent
Description: Constructors for the "finalDescendent" output analysis class.
Code lines: 3

Contained by: file nodes.property_extractor.final_descendent.F90

file: nodes.property_extractor.half_light_properties.F90
Description: Contains a module which implements a half-light radii property extractor class.
Code lines: 157

interface: nodepropertyextractorradiihalflightproperties
Description: Constructors for the "radiiHalfLightProperties" output analysis class.
Code lines: 3
Contained by: file nodes.property_extractor.half_light_properties.F90

function: radiihalflightpropertiesconstructorparameters
Description: Constructor for the radiiHalfLightProperties property extractor class which takes a parameter set as input.
Code lines: 10
Contained by: file nodes.property_extractor.half_light_properties.F90
Modules used: input_parameters

function: radiihalflightpropertiesdescriptions
Description: Return descriptions of the radiiHalfLightProperties property extractor class.
Code lines: 15
Contained by: file nodes.property_extractor.half_light_properties.F90
Modules used: stellar_luminosities_structure

function: radiihalflightpropertieselementcount
Description: Return the number of elements in the radiiHalfLightProperties property extractor class.
Code lines: 10
Contained by: file nodes.property_extractor.half_light_properties.F90
Modules used: stellar_luminosities_structure

function: radiihalflightpropertiesextract
Description: Implement a radiiHalfLightProperties property extractor.
Code lines: 26
Contained by: file nodes.property_extractor.half_light_properties.F90
Modules used: galactic_structure_enclosed_masses galactic_structure_options stellar_luminosities_structure

function: radiihalflightpropertiesnames
Description: Return the names of the radiiHalfLightProperties properties.
Code lines: 15
Contained by: file nodes.property_extractor.half_light_properties.F90
Modules used: stellar_luminosities_structure

function: radiihalflightpropertiestype
Description: Return the type of the last isolated redshift property.
Code lines: 9
Contained by: file nodes.property_extractor.half_light_properties.F90
Modules used: output_analyses_options

function: `radiihalflightpropertiesunitsinsi`
Description: Return the units of the last isolated redshift property in the SI system.
Code lines: 16
Contained by: file `nodes.property_extractor.half_light_properties.F90`
Modules used: `numerical_constants_astronomical` `stellar_luminosities_structure`

file: `nodes.property_extractor.half_mass_radius.F90`
Description: Contains a module which implements a radiusHalfMass property extractor class.
Code lines: 111

interface: `nodepropertyextractorradiushalfmass`
Description: Constructors for the "radiusHalfMass" output analysis class.
Code lines: 3
Contained by: file `nodes.property_extractor.half_mass_radius.F90`

function: `radiushalfmassconstructorparameters`
Description: Constructor for the `radiusHalfMass` property extractor class which takes a parameter set as input.
Code lines: 10
Contained by: file `nodes.property_extractor.half_mass_radius.F90`
Modules used: `input_parameters`

function: `radiushalfmassdescription`
Description: Return a description of the radiusHalfMass property.
Code lines: 9
Contained by: file `nodes.property_extractor.half_mass_radius.F90`

function: `radiushalfmassextract`
Description: Implement a last isolated redshift output analysis.
Code lines: 12
Contained by: file `nodes.property_extractor.half_mass_radius.F90`
Modules used: `galactic_structure_enclosed_masses` `galactic_structure_options`

function: `radiushalfmassname`
Description: Return the name of the last isolated redshift property.
Code lines: 9
Contained by: file `nodes.property_extractor.half_mass_radius.F90`

function: `radiushalfmasstype`
Description: Return the type of the last isolated redshift property.
Code lines: 9
Contained by: file `nodes.property_extractor.half_mass_radius.F90`
Modules used: `output_analyses_options`

function: `radiushalfmassunitsinsi`
Description: Return the units of the last isolated redshift property in the SI system.
Code lines: 9
Contained by: file `nodes.property_extractor.half_mass_radius.F90`

Modules used: numerical_constants_astronomical

file: nodes.property_extractor.halo_bias.F90
Description: Contains a module which implements an ISM mass output analysis property extractor class.
Code lines: 140
Modules used: dark_matter_halo_biases

function: halobiasconstructorinternal
Description: Internal constructor for the haloBias node property extractor class.
Code lines: 8
Contained by: file nodes.property_extractor.halo_bias.F90

function: halobiasconstructorparameters
Description: Constructor for the haloBias node property extractor class which takes a parameter set as input.
Code lines: 13
Contained by: file nodes.property_extractor.halo_bias.F90
Modules used: input_parameters

function: halobiasdescription
Description: Return a description of the haloBias property.
Code lines: 9
Contained by: file nodes.property_extractor.halo_bias.F90

subroutine: halobiasdestructor
Description: Destructor for the haloBias property extractor class.
Code lines: 7
Contained by: file nodes.property_extractor.halo_bias.F90

function: halobiasextract
Description: Implement a haloBias node property extractor.
Code lines: 15
Contained by: file nodes.property_extractor.halo_bias.F90

function: halobiasname
Description: Return the name of the haloBias property.
Code lines: 9
Contained by: file nodes.property_extractor.halo_bias.F90

function: halobiastype
Description: Return the type of the stellar mass property.
Code lines: 9
Contained by: file nodes.property_extractor.halo_bias.F90
Modules used: output_analyses_options

function: halobiasunitsinsi
Description: Return the units of the haloBias property in the SI system.
Code lines: 8
Contained by: file nodes.property_extractor.halo_bias.F90

interface: `nodepropertyextractorhalobias`
Description: Constructors for the "haloBias" output analysis class.
Code lines: 4
Contained by: file nodes.property_extractor.halo_bias.F90

file: `nodes.property_extractor.halo_environment.F90`
Description: Contains a module which implements a node property extractor class for halo environment.
Code lines: 157
Modules used: cosmological_density_field

function: `haloenvironmentconstructorinternal`
Description: Internal constructor for the `haloEnvironment` output analysis property extractor class.
Code lines: 8
Contained by: file nodes.property_extractor.halo_environment.F90

function: `haloenvironmentconstructorparameters`
Description: Constructor for the `haloEnvironment` node property extractor class which takes a parameter set as input.
Code lines: 13
Contained by: file nodes.property_extractor.halo_environment.F90
Modules used: input_parameters

function: `haloenvironmentdescriptions`
Description: Return a description of the `haloEnvironment` property.
Code lines: 11
Contained by: file nodes.property_extractor.halo_environment.F90

subroutine: `haloenvironmentdestructor`
Description: Destructor for the `haloEnvironment` output analysis property extractor class.
Code lines: 7
Contained by: file nodes.property_extractor.halo_environment.F90

function: `haloenvironmentelementcount`
Description: Return the number of elements in the `haloEnvironment` property extractor.
Code lines: 9
Contained by: file nodes.property_extractor.halo_environment.F90

function: `haloenvironmentextract`
Description: Implement extraction of halo environment properties.
Code lines: 13
Contained by: file nodes.property_extractor.halo_environment.F90

function: `haloenvironmentnames`
Description: Return the name of the `haloEnvironment` property.
Code lines: 11
Contained by: file nodes.property_extractor.halo_environment.F90

function: `haloenvironmenttype`
Description: Return the type of the `haloEnvironment` property.

Code lines: 9
Contained by: file nodes.property_extractor.halo_environment.F90
Modules used: output_analyses_options

function: haloenvironmentunitsinsi
Description: Return the units of the haloEnvironment property in the SI system.
Code lines: 11
Contained by: file nodes.property_extractor.halo_environment.F90

interface: nodepropertyextractorhaloenvironment
Description: Constructors for the "haloEnvironment" output analysis class.
Code lines: 4
Contained by: file nodes.property_extractor.halo_environment.F90

file: nodes.property_extractor.host_mass.F90
Description: Contains a module which implements a massHost property extractor class.
Code lines: 118

function: masshostconstructorparameters
Description: Constructor for the massHost property extractor class which takes a parameter set as input.
Code lines: 10
Contained by: file nodes.property_extractor.host_mass.F90
Modules used: input_parameters

function: masshostdescription
Description: Return a description of the host halo mass property.
Code lines: 9
Contained by: file nodes.property_extractor.host_mass.F90

function: masshostextract
Description: Implement a last isolated redshift output analysis.
Code lines: 19
Contained by: file nodes.property_extractor.host_mass.F90
Modules used: galacticus_nodes

function: masshostname
Description: Return the name of the host halo mass property.
Code lines: 9
Contained by: file nodes.property_extractor.host_mass.F90

function: masshosttype
Description: Return the type of the last isolated redshift property.
Code lines: 9
Contained by: file nodes.property_extractor.host_mass.F90
Modules used: output_analyses_options

function: masshostunitsinsi
Description: Return the units of the host halo mass in the SI system.
Code lines: 9

Contained by: file nodes.property_extractor.host_mass.F90
Modules used: numerical_constants_astronomical

interface: nodepropertyextractormasshost
Description: Constructors for the "massHost" output analysis class.
Code lines: 3
Contained by: file nodes.property_extractor.host_mass.F90

file: nodes.property_extractor.hosts.F90
Description: Contains a module which implements an ISM mass output analysis property extractor class.
Code lines: 132

function: indiceshostconstructorinternal
Description: Internal constructor for the indicesHost node property extractor class.
Code lines: 8
Contained by: file nodes.property_extractor.hosts.F90

function: indiceshostconstructorparameters
Description: Constructor for the indicesHost node property extractor class which takes a parameter set as input.
Code lines: 19
Contained by: file nodes.property_extractor.hosts.F90
Modules used: input_parameters

function: indiceshostdescription
Description: Return a description of the indicesHost property.
Code lines: 9
Contained by: file nodes.property_extractor.hosts.F90

function: indiceshostextract
Description: Implement a indicesHost node property extractor.
Code lines: 23
Contained by: file nodes.property_extractor.hosts.F90

function: indiceshostname
Description: Return the name of the indicesHost property.
Code lines: 9
Contained by: file nodes.property_extractor.hosts.F90

function: indiceshosttype
Description: Return the type of the stellar mass property.
Code lines: 9
Contained by: file nodes.property_extractor.hosts.F90
Modules used: output_analyses_options

interface: nodepropertyextractorindiceshost
Description: Constructors for the "indicesHost" output analysis class.
Code lines: 4
Contained by: file nodes.property_extractor.hosts.F90

file: `nodes.property_extractor.hot_mode_accretion_fraction.F90`
Description: Contains a module which implements a hot mode accretion fraction rate property extractor class.
Code lines: 144
Modules used: accretion_halos

function: `fractionaccretionhotmodeconstructorinternal`
Description: Internal constructor for the `fractionAccretionHotMode` property extractor class.
Code lines: 8
Contained by: file nodes.property_extractor.hot_mode_accretion_fraction.F90

function: `fractionaccretionhotmodeconstructorparameters`
Description: Constructor for the `fractionAccretionHotMode` property extractor class which takes a parameter set as input.
Code lines: 13
Contained by: file nodes.property_extractor.hot_mode_accretion_fraction.F90
Modules used: input_parameters

function: `fractionaccretionhotmodedescription`
Description: Return a description of the `fractionAccretionHotMode` property.
Code lines: 9
Contained by: file nodes.property_extractor.hot_mode_accretion_fraction.F90

subroutine: `fractionaccretionhotmodedestructor`
Description: Destructor for the `fractionAccretionHotMode` property extractor class.
Code lines: 7
Contained by: file nodes.property_extractor.hot_mode_accretion_fraction.F90

function: `fractionaccretionhotmodeextract`
Description: Implement a `fractionAccretionHotMode` property extractor.
Code lines: 18
Contained by: file nodes.property_extractor.hot_mode_accretion_fraction.F90
Modules used: accretion_halos

function: `fractionaccretionhotmodename`
Description: Return the name of the `fractionAccretionHotMode` property.
Code lines: 9
Contained by: file nodes.property_extractor.hot_mode_accretion_fraction.F90

function: `fractionaccretionhotmodetype`
Description: Return the type of the `fractionAccretionHotMode` property.
Code lines: 9
Contained by: file nodes.property_extractor.hot_mode_accretion_fraction.F90
Modules used: output_analyses_options

function: `fractionaccretionhotmodeunitsinsi`
Description: Return the units of the `fractionAccretionHotMode` property in the SI system.
Code lines: 8
Contained by: file nodes.property_extractor.hot_mode_accretion_fraction.F90

interface: `nodepropertyextractorfractionaccretionhotmode`
Description: Constructors for the `fractionAccretionHotMode` output analysis class.
Code lines: 4
Contained by: file nodes.property_extractor.hot_mode_accretion_fraction.F90

file: `nodes.property_extractor.integer_scalar.F90`
Code lines: 97
Modules used: kind_numbers

function: `integerscalarunitsinsi`
Description: Interface for integerScalar property units.
Code lines: 8
Contained by: file nodes.property_extractor.integer_scalar.F90

type: `nodepropertyextractorintegerscalar`
Description: A scalar integer node property extractor class.
Code lines: 39
Contained by: file nodes.property_extractor.integer_scalar.F90

file: `nodes.property_extractor.integer_tuple.F90`
Code lines: 113
Modules used: kind_numbers

type: `nodepropertyextractorintegertuple`
Description: A integerTuple property extractor.
Code lines: 47
Contained by: file nodes.property_extractor.integer_tuple.F90

file: `nodes.property_extractor.lightcone.F90`
Code lines: 281
Modules used: cosmology_functions geometry_lightcones iso_c_binding

subroutine: `lightconeaddinstances`
Description: Implement adding of instances to a lightcone property extractor.
Code lines: 25
Contained by: file nodes.property_extractor.lightcone.F90
Modules used: galacticus_error string_handling

function: `lightconeconstructorinternal`
Description: Internal constructor for the “lightcone” output extractor property extractor class.
Code lines: 61
Contained by: file nodes.property_extractor.lightcone.F90
Modules used: numerical_constants_astronomical numerical_constants_prefixes

function: `lightconeconstructorparameters`
Description: Constructor for the “lightcone” output extractor property extractor class which takes a parameter set as input.
Code lines: 33

Contained by: file nodes.property_extractor.lightcone.F90
Modules used: input_parameters

function: lightconedescriptions
Description: Return the descriptions of the lightconeple properties.
Code lines: 11
Contained by: file nodes.property_extractor.lightcone.F90

subroutine: lightconedestructor
Description: Destructor for the lightcone output extractor property extractor class.
Code lines: 8
Contained by: file nodes.property_extractor.lightcone.F90

function: lightconeelementcount
Description: Return the number of elements in the lightconeple property extractors.
Code lines: 9
Contained by: file nodes.property_extractor.lightcone.F90

function: lightconeextract
Description: Implement a lightcone output extractor.
Code lines: 29
Contained by: file nodes.property_extractor.lightcone.F90
Modules used: galacticus_error numerical_constants_astronomical numerical_constants_physical numerical_constants_prefixes vectors

function: lightconenames
Description: Return the names of the lightconeple properties.
Code lines: 11
Contained by: file nodes.property_extractor.lightcone.F90

function: lightconetype
Description: Return the type of the lightcone property.
Code lines: 9
Contained by: file nodes.property_extractor.lightcone.F90
Modules used: output_analyses_options

function: lightconeunitsinsi
Description: Return the units of the lightconeple properties in the SI system.
Code lines: 11
Contained by: file nodes.property_extractor.lightcone.F90

interface: nodepropertyextractorlightcone
Description: Constructors for the "lightcone" output extractor class.
Code lines: 4
Contained by: file nodes.property_extractor.lightcone.F90

file: nodes.property_extractor.luminosity_emission_line.F90
Description: Contains a module which implements a stellar mass output analysis property extractor class.

Code lines: 472
Modules used: fgsl, iso_varying_string, output_times, stellar_spectra_dust_attenuations

function: lmnstyemssnlineconstructorinternal
Description: Internal constructor for the "lmnstyEmssnLine" output analysis property extractor class.
Code lines: 76
Contained by: file nodes.property_extractor.luminosity_emission_line.F90
Modules used: galacticus_error, galacticus_paths, instruments_filters, io_hdf5, iso_c_binding, memory_management, output_times, stellar_luminosities_structure, string_handling

function: lmnstyemssnlineconstructorparameters
Description: Constructor for the "lmnstyEmssnLine" output analysis property extractor class which takes a parameter set as input.
Code lines: 34
Contained by: file nodes.property_extractor.luminosity_emission_line.F90
Modules used: input_parameters

function: lmnstyemssnlinedescription
Description: Return a description of the lmnstyEmssnLine property.
Code lines: 8
Contained by: file nodes.property_extractor.luminosity_emission_line.F90

subroutine: lmnstyemssnlinedestructor
Description: Destructor for the "lmnstyEmssnLine" output analysis property extractor class.
Code lines: 14
Contained by: file nodes.property_extractor.luminosity_emission_line.F90
Modules used: numerical_interpolation

function: lmnstyemssnlineextract
Description: Implement an emission line output analysis property extractor.
Code lines: 198
Contained by: file nodes.property_extractor.luminosity_emission_line.F90
Modules used: abundances_structure, galacticus_nodes, iso_c_binding, numerical_constants_astronomical, numerical_constants_atomic, numerical_constants_physical, numerical_constants_prefixes, numerical_interpolation, stellar_luminosities_structure

function: lmnstyemssnlinename
Description: Return the name of the lmnstyEmssnLine property.
Code lines: 8
Contained by: file nodes.property_extractor.luminosity_emission_line.F90

function: lmnstyemssnlinequantity
Description: Return the class of the emission line luminosity property.

Code lines: 9
Contained by: file nodes.property_extractor.luminosity_emission_line.F90
Modules used: output_analyses_options

function: lmnstyemssnlinetype
Description: Return the type of the emission line luminosity property.
Code lines: 9
Contained by: file nodes.property_extractor.luminosity_emission_line.F90
Modules used: output_analyses_options

function: lmnstyemssnlineunitsinsi
Description: Return the units of the lmnstyEmssnLine property in the SI system.
Code lines: 9
Contained by: file nodes.property_extractor.luminosity_emission_line.F90
Modules used: numerical_constants_units

interface: nodepropertyextractorlmnstyemssnline
Description: Constructors for the "lmnstyEmssnLine" output analysis class.
Code lines: 4
Contained by: file nodes.property_extractor.luminosity_emission_line.F90

file: nodes.property_extractor.luminosity_stellar.F90
Description: Contains a module which implements a stellar mass output analysis property extractor class.
Code lines: 236
Modules used: iso_varying_string output_times

function: luminositystellarconstructorinternal
Description: Internal constructor for the "luminosityStellar" output analysis property extractor class.
Code lines: 36
Contained by: file nodes.property_extractor.luminosity_stellar.F90
Modules used: iso_c_binding memory_management
stellar_luminosities_structure

function: luminositystellarconstructorparameters
Description: Constructor for the "luminosityStellar" output analysis property extractor class which takes a parameter set as input.
Code lines: 62
Contained by: file nodes.property_extractor.luminosity_stellar.F90
Modules used: input_parameters

function: luminositystellardescription
Description: Return a description of the luminosityStellar property.
Code lines: 8
Contained by: file nodes.property_extractor.luminosity_stellar.F90

subroutine: luminositystellardestructor
Description: Destructor for the "luminosityStellar" output analysis property extractor class.
Code lines: 8
Contained by: file nodes.property_extractor.luminosity_stellar.F90

Modules used: memory_management

function: luminositystellarextract
Description: Implement a stellar luminosity output analysis property extractor.
Code lines: 18
Contained by: file nodes.property_extractor.luminosity_stellar.F90
Modules used: galactic_structure_enclosed_masses galactic_structure_options galacticus_nodes iso_c_binding

function: luminositystellarname
Description: Return the name of the luminosityStellar property.
Code lines: 8
Contained by: file nodes.property_extractor.luminosity_stellar.F90

function: luminositystellarquantity
Description: Return the class of the stellar luminosity property.
Code lines: 9
Contained by: file nodes.property_extractor.luminosity_stellar.F90
Modules used: output_analyses_options

function: luminositystellartype
Description: Return the type of the stellar luminosity property.
Code lines: 9
Contained by: file nodes.property_extractor.luminosity_stellar.F90
Modules used: output_analyses_options

function: luminositystellarunitsinsi
Description: Return the units of the luminosityStellar property in the SI system.
Code lines: 9
Contained by: file nodes.property_extractor.luminosity_stellar.F90
Modules used: numerical_constants_astronomical

interface: nodepropertyextractorluminositystellar
Description: Constructors for the "luminosityStellar" output analysis class.
Code lines: 4
Contained by: file nodes.property_extractor.luminosity_stellar.F90

file: nodes.property_extractor.luminosity_stellar.dust.CharlotFall2000.F90
Description: Contains a module which implements a stellar luminosity output analysis property extractor class which applies the dust model of Charlot and Fall [2000].
Code lines: 292
Modules used: iso_varying_string output_times

function: lmnstystllrcf2000description
Description: Return a description of the lmnstyStllrCF2000 property.
Code lines: 8
Contained by: file nodes.property_extractor.luminosity_stellar.dust.CharlotFall2000.F90

function: lmnstystllrcf2000name

Description: Return the name of the lmnstyStllrCF2000 property.
Code lines: 8
Contained by: file nodes.property_extractor.luminosity_stellar.dust.CharlotFall2000.F90

function: `lmnstystllrcf2000unitsinsi`
Description: Return the units of the lmnstyStllrCF2000 property in the SI system.
Code lines: 9
Contained by: file nodes.property_extractor.luminosity_stellar.dust.CharlotFall2000.F90
Modules used: numerical_constants_astronomical

function: `lmnstystllrchrltfll2000constructorinternal`
Description: Internal constructor for the "lmnstyStllrChrltFll2000" output analysis property extractor class.
Code lines: 39
Contained by: file nodes.property_extractor.luminosity_stellar.dust.CharlotFall2000.F90
Modules used: instruments_filters, iso_c_binding, memory_management, stellar_luminosities_structure

function: `lmnstystllrchrltfll2000constructorparameters`
Description: Constructor for the "lmnstyStllrChrltFll2000" output analysis property extractor class which takes a parameter set as input.
Code lines: 68
Contained by: file nodes.property_extractor.luminosity_stellar.dust.CharlotFall2000.F90
Modules used: input_parameters

subroutine: `lmnstystllrchrltfll2000destructor`
Description: Destructor for the `lmnstyStllrChrltFll2000` output analysis property extractor class.
Code lines: 7
Contained by: file nodes.property_extractor.luminosity_stellar.dust.CharlotFall2000.F90

function: `lmnstystllrchrltfll2000extract`
Description: Implement a stellar luminosity output analysis property extractor.
Code lines: 66
Contained by: file nodes.property_extractor.luminosity_stellar.dust.CharlotFall2000.F90
Modules used: abundances_structure, galacticus_nodes, iso_c_binding, numerical_constants_astronomical, numerical_constants_atomic, numerical_constants_math, numerical_constants_prefixes, stellar_luminosities_structure

function: `lmnstystllrchrltfll2000quantity`
Description: Return the class of the stellar luminosity property.
Code lines: 9
Contained by: file nodes.property_extractor.luminosity_stellar.dust.CharlotFall2000.F90
Modules used: output_analyses_options

function: `lmnstystllrchrltfll2000type`
Description: Return the type of the stellar luminosity property.
Code lines: 9
Contained by: file nodes.property_extractor.luminosity_stellar.dust.CharlotFall2000.F90

Modules used: output_analyses_options

interface: nodepropertyextractorlmnstystllrcf2000
Description: Constructors for the "lmnstyStllrChrltFll2000" output analysis class.
Code lines: 4
Contained by: file nodes.property_extractor.luminosity_stellar.dust.CharlotFall2000.F90

file: nodes.property_extractor.main_branch.F90
Description: Contains a module which implements an ISM mass output analysis property extractor class.
Code lines: 103

function: mainbranchstatusconstructorparameters
Description: Constructor for the mainBranchStatus node property extractor class which takes a parameter set as input.
Code lines: 10
Contained by: file nodes.property_extractor.main_branch.F90
Modules used: input_parameters

function: mainbranchstatusdescription
Description: Return a description of the mainBranchStatus property.
Code lines: 9
Contained by: file nodes.property_extractor.main_branch.F90

function: mainbranchstatusextract
Description: Implement a mainBranchStatus node property extractor.
Code lines: 16
Contained by: file nodes.property_extractor.main_branch.F90

function: mainbranchstatusname
Description: Return the name of the mainBranchStatus property.
Code lines: 9
Contained by: file nodes.property_extractor.main_branch.F90

function: mainbranchstatustype
Description: Return the type of the stellar mass property.
Code lines: 9
Contained by: file nodes.property_extractor.main_branch.F90
Modules used: output_analyses_options

interface: nodepropertyextractormainbranchstatus
Description: Constructors for the "mainBranchStatus" output analysis class.
Code lines: 3
Contained by: file nodes.property_extractor.main_branch.F90

file: nodes.property_extractor.mass_ISM.F90
Description: Contains a module which implements an ISM mass output analysis property extractor class.
Code lines: 123

function: massismconstructorparameters

Description: Constructor for the "massISM" output analysis property extractor class which takes a parameter set as input.
Code lines: 10
Contained by: file nodes.property_extractor.mass_ISM.F90
Modules used: input_parameters

function: massismdescription
Description: Return a description of the massISM property.
Code lines: 9
Contained by: file nodes.property_extractor.mass_ISM.F90

function: massismextract
Description: Implement a massISM output analysis.
Code lines: 12
Contained by: file nodes.property_extractor.mass_ISM.F90
Modules used: galactic_structure_enclosed_masses galactic_structure_options

function: massismname
Description: Return the name of the massISM property.
Code lines: 9
Contained by: file nodes.property_extractor.mass_ISM.F90

function: massismquantity
Description: Return the class of the stellar luminosity property.
Code lines: 9
Contained by: file nodes.property_extractor.mass_ISM.F90
Modules used: output_analyses_options

function: massismtype
Description: Return the type of the stellar mass property.
Code lines: 9
Contained by: file nodes.property_extractor.mass_ISM.F90
Modules used: output_analyses_options

function: massismunitsinsi
Description: Return the units of the massISM property in the SI system.
Code lines: 9
Contained by: file nodes.property_extractor.mass_ISM.F90
Modules used: numerical_constants_astronomical

interface: nodepropertyextractormassism
Description: Constructors for the "massISM" output analysis class.
Code lines: 3
Contained by: file nodes.property_extractor.mass_ISM.F90

file: nodes.property_extractor.mass_black_hole.F90
Description: Contains a module which implements a black hole mass output analysis property extractor class.
Code lines: 126

function: `massblackholeconstructorparameters`
Description: Constructor for the "massBlackHole" output analysis property extractor class which takes a parameter set as input.
Code lines: 10
Contained by: file `nodes.property_extractor.mass_black_hole.F90`
Modules used: `input_parameters`

function: `massblackholedescription`
Description: Return a description of the massBlackHole property.
Code lines: 9
Contained by: file `nodes.property_extractor.mass_black_hole.F90`

function: `massblackholeextract`
Description: Implement a massBlackHole output analysis.
Code lines: 15
Contained by: file `nodes.property_extractor.mass_black_hole.F90`
Modules used: `galactic_structure_enclosed_masses` `galactic_structure_options` `galacticus_nodes`

function: `massblackholename`
Description: Return the name of the massBlackHole property.
Code lines: 9
Contained by: file `nodes.property_extractor.mass_black_hole.F90`

function: `massblackholequantity`
Description: Return the class of the stellar luminosity property.
Code lines: 9
Contained by: file `nodes.property_extractor.mass_black_hole.F90`
Modules used: `output_analyses_options`

function: `massblackholetype`
Description: Return the type of the stellar mass property.
Code lines: 9
Contained by: file `nodes.property_extractor.mass_black_hole.F90`
Modules used: `output_analyses_options`

function: `massblackholeunitsinsi`
Description: Return the units of the massBlackHole property in the SI system.
Code lines: 9
Contained by: file `nodes.property_extractor.mass_black_hole.F90`
Modules used: `numerical_constants_astronomical`

interface: `nodepropertyextractormassblackhole`
Description: Constructors for the "massBlackHole" output analysis class.
Code lines: 3
Contained by: file `nodes.property_extractor.mass_black_hole.F90`

file: `nodes.property_extractor.mass_halo.F90`

Description: Contains a module which implements a halo mass output analysis property extractor class.
Code lines: 144
Modules used: virial_density_contrast

function: masshaloconstructorinternal
Description: Internal constructor for the "massHalo" output analysis property extractor class.
Code lines: 9
Contained by: file nodes.property_extractor.mass_halo.F90
Modules used: input_parameters

function: masshaloconstructorparameters
Description: Constructor for the "massHalo" output analysis property extractor class which takes a parameter set as input.
Code lines: 13
Contained by: file nodes.property_extractor.mass_halo.F90
Modules used: input_parameters

function: masshalodescription
Description: Return a description of the massHalo property.
Code lines: 9
Contained by: file nodes.property_extractor.mass_halo.F90

subroutine: masshalodestructor
Description: Destructor for the mass output analysis property extractor class.
Code lines: 7
Contained by: file nodes.property_extractor.mass_halo.F90

function: masshaloextract
Description: Implement a massHalo output analysis.
Code lines: 14
Contained by: file nodes.property_extractor.mass_halo.F90
Modules used: dark_matter_profile_mass_definitions galacticus_nodes

function: masshaloname
Description: Return the name of the massHalo property.
Code lines: 9
Contained by: file nodes.property_extractor.mass_halo.F90

function: masshalotype
Description: Return the type of the halo mass property.
Code lines: 9
Contained by: file nodes.property_extractor.mass_halo.F90
Modules used: output_analyses_options

function: masshalounitsinsi
Description: Return the units of the massHalo property in the SI system.
Code lines: 9
Contained by: file nodes.property_extractor.mass_halo.F90
Modules used: numerical_constants_astronomical

interface: `nodepropertyextractormasshalo`
Description: Constructors for the "massHalo" output analysis class.
Code lines: 4
Contained by: file `nodes.property_extractor.mass_halo.F90`

file: `nodes.property_extractor.mass_profile.F90`
Description: Contains a module which implements a property extractor class for the mass enclosed by a set of radii.
Code lines: 173

function: `massprofileconstructorinternal`
Description: Internal constructor for the `massProfile` property extractor class.
Code lines: 10
Contained by: file `nodes.property_extractor.mass_profile.F90`
Modules used: `galactic_structure_options`

function: `massprofileconstructorparameters`
Description: Constructor for the `massProfile` property extractor class which takes a parameter set as input.
Code lines: 19
Contained by: file `nodes.property_extractor.mass_profile.F90`
Modules used: `input_parameters`

function: `massprofiledescriptions`
Description: Return descriptions of the `massProfile` property.
Code lines: 16
Contained by: file `nodes.property_extractor.mass_profile.F90`

function: `massprofileelementcount`
Description: Return the number of elements in the `massProfile` property extractors.
Code lines: 9
Contained by: file `nodes.property_extractor.mass_profile.F90`

function: `massprofileextract`
Description: Implement a last isolated redshift output analysis.
Code lines: 18
Contained by: file `nodes.property_extractor.mass_profile.F90`
Modules used: `galactic_structure_enclosed_masses` `galactic_structure_options`

function: `massprofilenames`
Description: Return the names of the `massProfile` properties.
Code lines: 16
Contained by: file `nodes.property_extractor.mass_profile.F90`

function: `massprofiletype`
Description: Return the type of the `massProfile` properties.
Code lines: 9
Contained by: file `nodes.property_extractor.mass_profile.F90`

Modules used: output_analyses_options

function: massprofileunitsinsi
Description: Return the units of the massProfile properties in the SI system.
Code lines: 15
Contained by: file nodes.property_extractor.mass_profile.F90
Modules used: numerical_constants_astronomical

interface: nodepropertyextractormassprofile
Description: Constructors for the "massProfile" output analysis class.
Code lines: 4
Contained by: file nodes.property_extractor.mass_profile.F90

file: nodes.property_extractor.mass_stellar.F90
Description: Contains a module which implements a stellar mass output analysis property extractor class.
Code lines: 123

function: massstellarconstructorparameters
Description: Constructor for the "massStellar" output analysis property extractor class which takes a parameter set as input.
Code lines: 10
Contained by: file nodes.property_extractor.mass_stellar.F90
Modules used: input_parameters

function: massstellardescription
Description: Return a description of the massStellar property.
Code lines: 9
Contained by: file nodes.property_extractor.mass_stellar.F90

function: massstellarextract
Description: Implement a massStellar output analysis.
Code lines: 12
Contained by: file nodes.property_extractor.mass_stellar.F90
Modules used: galactic_structure_enclosed_masses galactic_structure_options

function: massstellarname
Description: Return the name of the massStellar property.
Code lines: 9
Contained by: file nodes.property_extractor.mass_stellar.F90

function: massstellarquantity
Description: Return the class of the stellar luminosity property.
Code lines: 9
Contained by: file nodes.property_extractor.mass_stellar.F90
Modules used: output_analyses_options

function: massstellartype
Description: Return the type of the stellar mass property.
Code lines: 9

Contained by: file nodes.property_extractor.mass_stellar.F90
Modules used: output_analyses_options

function: massstellarunitsinsi
Description: Return the units of the massStellar property in the SI system.
Code lines: 9
Contained by: file nodes.property_extractor.mass_stellar.F90
Modules used: numerical_constants_astronomical

interface: nodepropertyextractormassstellar
Description: Constructors for the "massStellar" output analysis class.
Code lines: 3
Contained by: file nodes.property_extractor.mass_stellar.F90

file: nodes.property_extractor.mass_stellar_morphology.F90
Description: Contains a module which implements a stellar mass-weighted morphology output analysis property extractor class.
Code lines: 117

function: massstellarmorphologyconstructorparameters
Description: Constructor for the "massStellarMorphology" output analysis property extractor class which takes a parameter set as input.
Code lines: 10
Contained by: file nodes.property_extractor.mass_stellar_morphology.F90
Modules used: input_parameters

function: massstellarmorphologydescription
Description: Return a description of the massStellarMorphology property.
Code lines: 9
Contained by: file nodes.property_extractor.mass_stellar_morphology.F90

function: massstellarmorphologyextract
Description: Implement a stellar mass-weighted morphology output analysis.
Code lines: 19
Contained by: file nodes.property_extractor.mass_stellar_morphology.F90
Modules used: galactic_structure_enclosed_masses galactic_structure_options

function: massstellarmorphologyname
Description: Return the name of the massStellarMorphology property.
Code lines: 9
Contained by: file nodes.property_extractor.mass_stellar_morphology.F90

function: massstellarmorphologytype
Description: Return the type of the stellar mass-weighted morphology property.
Code lines: 9
Contained by: file nodes.property_extractor.mass_stellar_morphology.F90
Modules used: output_analyses_options

function: massstellarmorphologyunitsinsi

Description: Return the units of the massStellarMorphology property in the SI system.
Code lines: 8
Contained by: file nodes.property_extractor.mass_stellar_morphology.F90

interface: `nodepropertyextractormassstellarmorphology`
Description: Constructors for the "massStellarMorphology" output analysis class.
Code lines: 3
Contained by: file nodes.property_extractor.mass_stellar_morphology.F90

file: `nodes.property_extractor.mass_stellar_spheroid.F90`
Description: Contains a module which implements a spheroid stellar mass output analysis property extractor class.
Code lines: 123

function: `massstellarspheroidconstructorparameters`
Description: Constructor for the "massStellarSpheroid" output analysis property extractor class which takes a parameter set as input.
Code lines: 10
Contained by: file nodes.property_extractor.mass_stellar_spheroid.F90
Modules used: input_parameters

function: `massstellarspheroiddescription`
Description: Return a description of the massStellarSpheroid property.
Code lines: 9
Contained by: file nodes.property_extractor.mass_stellar_spheroid.F90

function: `massstellarspheroidextract`
Description: Implement a stellar mass-weighted morphology output analysis.
Code lines: 12
Contained by: file nodes.property_extractor.mass_stellar_spheroid.F90
Modules used: galactic_structure_enclosed_masses galactic_structure_options

function: `massstellarspheroidname`
Description: Return the name of the massStellarSpheroid property.
Code lines: 9
Contained by: file nodes.property_extractor.mass_stellar_spheroid.F90

function: `massstellarspheroidquantity`
Description: Return the class of the stellar luminosity property.
Code lines: 9
Contained by: file nodes.property_extractor.mass_stellar_spheroid.F90
Modules used: output_analyses_options

function: `massstellarspheroidtype`
Description: Return the type of the stellar mass-weighted morphology property.
Code lines: 9
Contained by: file nodes.property_extractor.mass_stellar_spheroid.F90
Modules used: output_analyses_options

function: `massstellarspheroidunitsinsi`
Description: Return the units of the massStellarSpheroid property in the SI system.
Code lines: 9
Contained by: file `nodes.property_extractor.mass_stellar_spheroid.F90`
Modules used: `numerical_constants_astronomical`

interface: `nodepropertyextractormassstellarspheroid`
Description: Constructors for the "massStellarSpheroid" output analysis class.
Code lines: 3
Contained by: file `nodes.property_extractor.mass_stellar_spheroid.F90`

file: `nodes.property_extractor.metallicity_ISM.F90`
Description: Contains a module which implements an ISM metallicity output analysis property extractor class.
Code lines: 153

function: `metallicityismconstructorinternal`
Description: Internal constructor for the "metallicityISM" output analysis property extractor class.
Code lines: 8
Contained by: file `nodes.property_extractor.metallicity_ISM.F90`

function: `metallicityismconstructorparameters`
Description: Constructor for the "metallicityISM" output analysis property extractor class which takes a parameter set as input.
Code lines: 21
Contained by: file `nodes.property_extractor.metallicity_ISM.F90`
Modules used: `abundances_structure` `input_parameters`

function: `metallicityismdescription`
Description: Return a description of the metallicityISM property.
Code lines: 10
Contained by: file `nodes.property_extractor.metallicity_ISM.F90`
Modules used: `abundances_structure`

function: `metallicityismextract`
Description: Extracts the metallicity (defined as the mass ratio of a specified element to hydrogen) in the ISM.
Code lines: 30
Contained by: file `nodes.property_extractor.metallicity_ISM.F90`
Modules used: `abundances_structure` `galacticus_nodes`

function: `metallicityismname`
Description: Return the name of the metallicityISM property.
Code lines: 10
Contained by: file `nodes.property_extractor.metallicity_ISM.F90`
Modules used: `abundances_structure`

function: `metallicityismtype`

Description: Return the type of the stellar mass property.
Code lines: 9
Contained by: file nodes.property_extractor.metallicity_ISM.F90
Modules used: output_analyses_options

function: metallicityismunitsinsi
Description: Return the units of the metallicityISM property in the SI system.
Code lines: 8
Contained by: file nodes.property_extractor.metallicity_ISM.F90

interface: nodepropertyextractormetallicityism
Description: Constructors for the "metallicityISM" output analysis class.
Code lines: 4
Contained by: file nodes.property_extractor.metallicity_ISM.F90

file: nodes.property_extractor.most_massive_progenitor.F90
Description: Contains a module which implements an ISM mass output analysis property extractor class.
Code lines: 144

function: mostmassiveprogenitorconstructorinternal
Description: Internal constructor for the mostMassiveProgenitor node property extractor class.
Code lines: 9
Contained by: file nodes.property_extractor.most_massive_progenitor.F90

function: mostmassiveprogenitorconstructorparameters
Description: Constructor for the mostMassiveProgenitor node property extractor class which takes a parameter set as input.
Code lines: 10
Contained by: file nodes.property_extractor.most_massive_progenitor.F90
Modules used: input_parameters

function: mostmassiveprogenitordescription
Description: Return a description of the mostMassiveProgenitor property.
Code lines: 9
Contained by: file nodes.property_extractor.most_massive_progenitor.F90

function: mostmassiveprogenitorextract
Description: Implement a mostMassiveProgenitor node property extractor.
Code lines: 43
Contained by: file nodes.property_extractor.most_massive_progenitor.F90
Modules used: galacticus_nodes merger_tree_walkers

function: mostmassiveprogenitorname
Description: Return the name of the mostMassiveProgenitor property.
Code lines: 9
Contained by: file nodes.property_extractor.most_massive_progenitor.F90

function: mostmassiveprogenitortype
Description: Return the type of the stellar mass property.

Code lines: 9
Contained by: file nodes.property_extractor.most_massive_progenitor.F90
Modules used: output_analyses_options

interface: nodepropertyextractormostmassiveprogenitor
Description: Constructors for the "mostMassiveProgenitor" output analysis class.
Code lines: 4
Contained by: file nodes.property_extractor.most_massive_progenitor.F90

file: nodes.property_extractor.multi.F90
Description: Contains a module which implements a multi node property extractor class.
Code lines: 474

subroutine: multiaddinstances
Description: Implement adding of instances to a multi output extractor.
Code lines: 14
Contained by: file nodes.property_extractor.multi.F90

function: multiconstructorinternal
Description: Internal constructor for the "multi" output extractor property extractor class.
Code lines: 14
Contained by: file nodes.property_extractor.multi.F90

function: multiconstructorparameters
Description: Constructor for the "multi" output extractor property extractor class which takes a parameter set as input.
Code lines: 22
Contained by: file nodes.property_extractor.multi.F90
Modules used: input_parameters

subroutine: multideepcopy
Description: Perform a deep copy for the multi extractor class.
Code lines: 32
Contained by: file nodes.property_extractor.multi.F90
Modules used: galacticus_error

function: multidescriptions
Description: Return the descriptions of the multiple properties.
Code lines: 44
Contained by: file nodes.property_extractor.multi.F90
Modules used: galacticus_error

subroutine: multidestructor
Description: Destructor for the multi output extractor property extractor class.
Code lines: 16
Contained by: file nodes.property_extractor.multi.F90

function: multielementcount
Description: Return the number of elements in the multiple property extractors.

Code lines: 27
Contained by: file nodes.property_extractor.multi.F90
Modules used: galacticus_error

function: multiextractdouble
Description: Implement a multi output extractor.
Code lines: 35
Contained by: file nodes.property_extractor.multi.F90
Modules used: galacticus_error

function: multiextractinteger
Description: Implement a multi output extractor.
Code lines: 35
Contained by: file nodes.property_extractor.multi.F90
Modules used: galacticus_error

type: multiextractorlist
Code lines: 3
Contained by: file nodes.property_extractor.multi.F90

function: multinames
Description: Return the names of the multiple properties.
Code lines: 44
Contained by: file nodes.property_extractor.multi.F90
Modules used: galacticus_error

function: multitype
Description: Return the type of the multi property.
Code lines: 9
Contained by: file nodes.property_extractor.multi.F90
Modules used: output_analyses_options

function: multiunitsinsi
Description: Return the units of the multiple properties in the SI system.
Code lines: 44
Contained by: file nodes.property_extractor.multi.F90
Modules used: galacticus_error

interface: nodepropertyextractormulti
Description: Constructors for the "multi" output extractor class.
Code lines: 4
Contained by: file nodes.property_extractor.multi.F90

file: nodes.property_extractor.node_indices.F90
Description: Contains a module which implements a property extractor for basic node indices.
Code lines: 136

function: nodeindicesconstructorparameters
Description: Constructor for the nodeIndices property extractor class which takes a parameter set as input.

Code lines: 10
Contained by: file nodes.property_extractor.node_indices.F90
Modules used: input_parameters

function: nodeindicesdescriptions
Description: Return descriptions of the nodeIndices properties.
Code lines: 11
Contained by: file nodes.property_extractor.node_indices.F90

function: nodeindiceselementcount
Description: Return the number of elements in the nodeIndices property extractors.
Code lines: 9
Contained by: file nodes.property_extractor.node_indices.F90

function: nodeindicesextract
Description: Implement a nodeIndices property extractor.
Code lines: 19
Contained by: file nodes.property_extractor.node_indices.F90

function: nodeindicesnames
Description: Return the names of the nodeIndices properties.
Code lines: 11
Contained by: file nodes.property_extractor.node_indices.F90

function: nodeindicestype
Description: Return the type of the last isolated redshift property.
Code lines: 9
Contained by: file nodes.property_extractor.node_indices.F90
Modules used: output_analyses_options

function: nodeindicesunitsinsi
Description: Return the units of the last isolated redshift property in the SI system.
Code lines: 11
Contained by: file nodes.property_extractor.node_indices.F90

interface: nodepropertyextractornodeindices
Description: Constructors for the "nodeIndices" output analysis class.
Code lines: 3
Contained by: file nodes.property_extractor.node_indices.F90

file: nodes.property_extractor.null.F90
Description: Contains a module which implements a null output analysis class.
Code lines: 60

interface: nodepropertyextractornull
Description: Constructors for the "null" output analysis class.
Code lines: 3
Contained by: file nodes.property_extractor.null.F90

function: `nullconstructorparameters`
Description: Constructor for the "null" output analysis property extractor class which takes a parameter set as input.
Code lines: 10
Contained by: file `nodes.property_extractor.null.F90`
Modules used: `input_parameters`

function: `nulltype`
Description: Return the type of the null property.
Code lines: 9
Contained by: file `nodes.property_extractor.null.F90`
Modules used: `output_analyses_options`

file: `nodes.property_extractor.projected_density.F90`
Description: Contains a module which implements a property extractor class for the projected density at a set of radii.
Code lines: 268
Modules used: `dark_matter_halo_scales` `galactic_structure_radii_definitions`

interface: `nodepropertyextractorprojecteddensity`
Description: Constructors for the "projectedDensity" output analysis class.
Code lines: 4
Contained by: file `nodes.property_extractor.projected_density.F90`

function: `projecteddensityconstructorinternal`
Description: Internal constructor for the `projectedDensity` property extractor class.
Code lines: 19
Contained by: file `nodes.property_extractor.projected_density.F90`
Modules used: `galactic_structure_radii_definitions`

function: `projecteddensityconstructorparameters`
Description: Constructor for the `projectedDensity` property extractor class which takes a parameter set as input.
Code lines: 31
Contained by: file `nodes.property_extractor.projected_density.F90`
Modules used: `input_parameters`

function: `projecteddensitydescriptions`
Description: Return descriptions of the `projectedDensity` property.
Code lines: 15
Contained by: file `nodes.property_extractor.projected_density.F90`

subroutine: `projecteddensitydestructor`
Description: Destructor for the `projectedDensity` property extractor class.
Code lines: 7
Contained by: file `nodes.property_extractor.projected_density.F90`

function: `projecteddensityelementcount`

Description: Return the number of elements in the `projectedDensity` property extractors.
Code lines: 9
Contained by: file nodes.property_extractor.projected_density.F90

function: `projecteddensityextract`
Description: Implement a `projectedDensity` property extractor.
Code lines: 73
Contained by: file nodes.property_extractor.projected_density.F90
Modules used: fgsl, galactic_structure_densities, galactic_structure_enclosed_masses, galactic_structure_options, galactic_structure_radii_definitions, galacticus_nodes, numerical_integration

function: `projecteddensityintegrand`
Description: Integrand function used for computing projected densities.
Code lines: 12
Contained by: function projecteddensityextract
Modules used: galactic_structure_densities

function: `projecteddensitynames`
Description: Return the names of the `projectedDensity` properties.
Code lines: 15
Contained by: file nodes.property_extractor.projected_density.F90

function: `projecteddensitytype`
Description: Return the type of the `projectedDensity` properties.
Code lines: 9
Contained by: file nodes.property_extractor.projected_density.F90
Modules used: output_analyses_options

function: `projecteddensityunitsinsi`
Description: Return the units of the `projectedDensity` properties in the SI system.
Code lines: 16
Contained by: file nodes.property_extractor.projected_density.F90
Modules used: numerical_constants_astronomical

file: `nodes.property_extractor.radius_half_mass.F90`
Description: Contains a module which implements a half-stellar mass radius output analysis property extractor class.
Code lines: 113

function: `halfmassradiusconstructorparameters`
Description: Constructor for the "halfMassRadius" output analysis property extractor class which takes a parameter set as input.
Code lines: 11
Contained by: file nodes.property_extractor.radius_half_mass.F90
Modules used: input_parameters

function: `halfmassradiusdescription`

Description: Return a description of the halfMassRadius property.
Code lines: 9
Contained by: file nodes.property_extractor.radius_half_mass.F90

function: halfmassradiusextract
Description: Implement a half-mass output analysis.
Code lines: 12
Contained by: file nodes.property_extractor.radius_half_mass.F90
Modules used: galactic_structure_enclosed_masses galactic_structure_options

function: halfmassradiusname
Description: Return the name of the halfMassRadius property.
Code lines: 9
Contained by: file nodes.property_extractor.radius_half_mass.F90

function: halfmassradiustype
Description: Return the type of the half-mass radius property.
Code lines: 9
Contained by: file nodes.property_extractor.radius_half_mass.F90
Modules used: output_analyses_options

function: halfmassradiusunitsinsi
Description: Return the units of the halfMassRadius property in the SI system.
Code lines: 9
Contained by: file nodes.property_extractor.radius_half_mass.F90
Modules used: numerical_constants_astronomical

interface: nodepropertyextractorhalfmassradius
Description: Constructors for the "halfMassRadius" output analysis class.
Code lines: 3
Contained by: file nodes.property_extractor.radius_half_mass.F90

file: nodes.property_extractor.ratio.F90
Description: Contains a module which implements a ratio output analysis property extractor class.
Code lines: 185

interface: nodepropertyextractorratio
Description: Constructors for the "ratio" output analysis class.
Code lines: 4
Contained by: file nodes.property_extractor.ratio.F90

function: ratioconstructorinternal
Description: Internal constructor for the "ratio" output analysis property extractor class.
Code lines: 24
Contained by: file nodes.property_extractor.ratio.F90
Modules used: galacticus_error

function: ratioconstructorparameters
Description: Constructor for the "ratio" output analysis property extractor class which takes a parameter set as input.
Code lines: 30

Contained by: file nodes.property_extractor.ratio.F90
Modules used: input_parameters

function: ratiodescription
Description: Return the description of this property.
Code lines: 8
Contained by: file nodes.property_extractor.ratio.F90

subroutine: ratiodestructor
Description: Destructor for the "ratio" output analysis property extractor class.
Code lines: 8
Contained by: file nodes.property_extractor.ratio.F90

function: ratioextract
Description: Implement a ratio output analysis.
Code lines: 29
Contained by: file nodes.property_extractor.ratio.F90
Modules used: galacticus_error

function: rationame
Description: Return the name of this property.
Code lines: 8
Contained by: file nodes.property_extractor.ratio.F90

function: ratiotype
Description: Return the type of the ratio property.
Code lines: 9
Contained by: file nodes.property_extractor.ratio.F90
Modules used: output_analyses_options

function: ratiounitsinsi
Description: Return the description of this property.
Code lines: 7
Contained by: file nodes.property_extractor.ratio.F90

file: nodes.property_extractor.redshift.F90
Description: Contains a module which implements a redshiftLastIsolated property extractor class.
Code lines: 138
Modules used: cosmology_functions

interface: nodepropertyextractorredshiftlastisolated
Description: Constructors for the "redshiftLastIsolated" output analysis class.
Code lines: 4
Contained by: file nodes.property_extractor.redshift.F90

function: redshiftlastisolatedconstructorinternal
Description: Internal constructor for the redshiftLastIsolated property extractor class.
Code lines: 8
Contained by: file nodes.property_extractor.redshift.F90

function: `redshiftlastisolatedconstructorparameters`
Description: Constructor for the `redshiftLastIsolated` property extractor class which takes a parameter set as input.
Code lines: 13
Contained by: file `nodes.property_extractor.redshift.F90`
Modules used: `input_parameters`

function: `redshiftlastisolateddescription`
Description: Return a description of the redshiftLastIsolated property.
Code lines: 9
Contained by: file `nodes.property_extractor.redshift.F90`

subroutine: `redshiftlastisolateddestructor`
Description: Destructor for the `redshiftLastIsolated` property extractor class.
Code lines: 7
Contained by: file `nodes.property_extractor.redshift.F90`

function: `redshiftlastisolatedextract`
Description: Implement a last isolated redshift output analysis.
Code lines: 13
Contained by: file `nodes.property_extractor.redshift.F90`
Modules used: `galacticus_nodes`

function: `redshiftlastisolatedname`
Description: Return the name of the last isolated redshift property.
Code lines: 9
Contained by: file `nodes.property_extractor.redshift.F90`

function: `redshiftlastisolatedtype`
Description: Return the type of the last isolated redshift property.
Code lines: 9
Contained by: file `nodes.property_extractor.redshift.F90`
Modules used: `output_analyses_options`

function: `redshiftlastisolatedunitsinsi`
Description: Return the units of the last isolated redshift property in the SI system.
Code lines: 8
Contained by: file `nodes.property_extractor.redshift.F90`

file: `nodes.property_extractor.rotation_curve.F90`
Description: Contains a module which implements a property extractor class for the rotation curve at a set of radii.
Code lines: 242
Modules used: `dark_matter_halo_scales` `galactic_structure_radii_definitions`

interface: `nodepropertyextractorrotationcurve`
Description: Constructors for the "rotationCurve" output analysis class.
Code lines: 4
Contained by: file `nodes.property_extractor.rotation_curve.F90`

function: `rotationcurveconstructorinternal`
Description: Internal constructor for the `rotationCurve` property extractor class.
Code lines: 19
Contained by: file nodes.property_extractor.rotation_curve.F90
Modules used: galactic_structure_radii_definitions

function: `rotationcurveconstructorparameters`
Description: Constructor for the `rotationCurve` property extractor class which takes a parameter set as input.
Code lines: 31
Contained by: file nodes.property_extractor.rotation_curve.F90
Modules used: input_parameters

function: `rotationcurvedescriptions`
Description: Return descriptions of the `rotationCurve` property.
Code lines: 15
Contained by: file nodes.property_extractor.rotation_curve.F90

subroutine: `rotationcurvedestructor`
Description: Destructor for the `rotationCurve` property extractor class.
Code lines: 7
Contained by: file nodes.property_extractor.rotation_curve.F90

function: `rotationcurveelementcount`
Description: Return the number of elements in the `rotationCurve` property extractors.
Code lines: 9
Contained by: file nodes.property_extractor.rotation_curve.F90

function: `rotationcurveextract`
Description: Implement a `rotationCurve` property extractor.
Code lines: 50
Contained by: file nodes.property_extractor.rotation_curve.F90
Modules used: galactic_structure_enclosed_masses galactic_structure_options galactic_structure_radii_definitions galactic_structure_rotation_curves galacticus_nodes

function: `rotationcurvenames`
Description: Return the names of the `rotationCurve` properties.
Code lines: 15
Contained by: file nodes.property_extractor.rotation_curve.F90

function: `rotationcurvetype`
Description: Return the type of the `rotationCurve` properties.
Code lines: 9
Contained by: file nodes.property_extractor.rotation_curve.F90
Modules used: output_analyses_options

function: `rotationcurveunitsinsi`
Description: Return the units of the `rotationCurve` properties in the SI system.

Code lines: 17
Contained by: file nodes.property_extractor.rotation_curve.F90
Modules used: numerical_constants_astronomical numerical_constants_prefixes

file: nodes.property_extractor.satellite_orbital_extrema.F90
Description: Contains a module which implements satellite orbital extrema property extractor class.
Code lines: 206

interface: nodepropertyextractorsatelliteorbitalextrema
Description: Constructors for the "satelliteOrbitalExtrema" output analysis class.
Code lines: 4
Contained by: file nodes.property_extractor.satellite_orbital_extrema.F90

function: satelliteorbitalextremaconstructorinternal
Description: Internal constructor for the satelliteOrbitalExtrema property extractor class.
Code lines: 19
Contained by: file nodes.property_extractor.satellite_orbital_extrema.F90
Modules used: galacticus_error

function: satelliteorbitalextremaconstructorparameters
Description: Constructor for the satelliteOrbitalExtrema property extractor class which takes a parameter set as input.
Code lines: 27
Contained by: file nodes.property_extractor.satellite_orbital_extrema.F90
Modules used: input_parameters

function: satelliteorbitalextremadescriptions
Description: Return a description of the satelliteOrbitalExtrema property.
Code lines: 12
Contained by: file nodes.property_extractor.satellite_orbital_extrema.F90

function: satelliteorbitalextremaelementcount
Description: Return the number of elements in the satelliteOrbitalExtrema property extractor class.
Code lines: 9
Contained by: file nodes.property_extractor.satellite_orbital_extrema.F90

function: satelliteorbitalextremaextract
Description: Implement a satelliteOrbitalExtrema property extractor
Code lines: 43
Contained by: file nodes.property_extractor.satellite_orbital_extrema.F90
Modules used: galacticus_nodes kepler_orbits satellite_orbits

function: satelliteorbitalextremanames
Description: Return the name of the satelliteOrbitalExtrema property.
Code lines: 12
Contained by: file nodes.property_extractor.satellite_orbital_extrema.F90

function: satelliteorbitalextrematype
Description: Return the type of the satelliteOrbitalExtrema property.

Code lines: 9
Contained by: file nodes.property_extractor.satellite_orbital_extrema.F90
Modules used: output_analyses_options

function: satelliteorbitalextremaunitsinsi
Description: Return the units of the satelliteOrbitalExtrema property in the SI system.
Code lines: 14
Contained by: file nodes.property_extractor.satellite_orbital_extrema.F90
Modules used: numerical_constants_astronomical numerical_constants_prefixes

file: nodes.property_extractor.satellite_status.F90
Description: Contains a module which implements an ISM mass output analysis property extractor class.
Code lines: 170

interface: nodepropertyextractorsatellitestatus
Description: Constructors for the "satelliteStatus" output analysis class.
Code lines: 4
Contained by: file nodes.property_extractor.satellite_status.F90

function: satellitestatusconstructorinternal
Description: Internal constructor for the satelliteStatus node property extractor class.
Code lines: 16
Contained by: file nodes.property_extractor.satellite_status.F90
Modules used: galacticus_error galacticus_nodes

function: satellitestatusconstructorparameters
Description: Constructor for the satelliteStatus node property extractor class which takes a parameter set as input.
Code lines: 19
Contained by: file nodes.property_extractor.satellite_status.F90
Modules used: input_parameters

function: satellitestatusdescription
Description: Return a description of the satelliteStatus property.
Code lines: 9
Contained by: file nodes.property_extractor.satellite_status.F90

function: satellitestatusextract
Description: Implement a satelliteStatus node property extractor.
Code lines: 43
Contained by: file nodes.property_extractor.satellite_status.F90
Modules used: galacticus_nodes histories

function: satellitestatusname
Description: Return the name of the satelliteStatus property.
Code lines: 9
Contained by: file nodes.property_extractor.satellite_status.F90

function: satellitestatustype
Description: Return the type of the stellar mass property.

Code lines: 9
Contained by: file nodes.property_extractor.satellite_status.F90
Modules used: output_analyses_options

file: nodes.property_extractor.scalar.F90
Code lines: 89

type: nodepropertyextractorscalar
Description: A scalar output analysis class.
Code lines: 39
Contained by: file nodes.property_extractor.scalar.F90

file: nodes.property_extractor.spin_Bullock.F90
Description: Contains a module which implements a node property extractor class for the Bullock et al. [2001] definition of spin parameter.
Code lines: 188
Modules used: dark_matter_halo_scales dark_matter_profiles_dmo

interface: nodepropertyextractorspinbullock
Description: Constructors for the "spinBullock" output analysis class.
Code lines: 4
Contained by: file nodes.property_extractor.spin_Bullock.F90

function: spinbullockconstructorinternal
Description: Internal constructor for the spinBullock output analysis property extractor class.
Code lines: 16
Contained by: file nodes.property_extractor.spin_Bullock.F90
Modules used: galacticus_nodes

function: spinbullockconstructorparameters
Description: Constructor for the y output analysis property extractor class which takes a parameter set as input.
Code lines: 16
Contained by: file nodes.property_extractor.spin_Bullock.F90
Modules used: input_parameters

function: spinbullockdescriptions
Description: Return a description of the spinBullock property.
Code lines: 12
Contained by: file nodes.property_extractor.spin_Bullock.F90

subroutine: spinbullockdestructor
Description: Destructor for the spinBullock output analysis property extractor class.
Code lines: 8
Contained by: file nodes.property_extractor.spin_Bullock.F90

function: spinbullockelementcount
Description: Return the number of elements in the spinBullock property extractor.
Code lines: 9

Contained by: file nodes.property_extractor.spin_Bullock.F90

function: spinbullockextract
Description: Implement extraction of the spin parameter under the Bullock et al. [2001] defintiion.
Code lines: 26
Contained by: file nodes.property_extractor.spin_Bullock.F90
Modules used: dark_matter_halo_spins galacticus_nodes

function: spinbullocknames
Description: Return the name of the spinBullock property.
Code lines: 12
Contained by: file nodes.property_extractor.spin_Bullock.F90

function: spinbullocktype
Description: Return the type of the spinBullock property.
Code lines: 9
Contained by: file nodes.property_extractor.spin_Bullock.F90
Modules used: output_analyses_options

function: spinbullockunitsinsi
Description: Return the units of the spinBullock property in the SI system.
Code lines: 11
Contained by: file nodes.property_extractor.spin_Bullock.F90

file: nodes.property_extractor.spin_parameter.F90
Description: Contains a module which implements a spin parameter output analysis property extractor class.
Code lines: 112

interface: nodepropertyextractorspin
Description: Constructors for the "spin" output property extractor class.
Code lines: 3
Contained by: file nodes.property_extractor.spin_parameter.F90

function: spinconstructorparameters
Description: Constructor for the "spin" output analysis property extractor class which takes a parameter set as input.
Code lines: 11
Contained by: file nodes.property_extractor.spin_parameter.F90
Modules used: input_parameters

function: spindescription
Description: Return a description of the spin property.
Code lines: 9
Contained by: file nodes.property_extractor.spin_parameter.F90

function: spinextract
Description: Implement a spin output property extractor.
Code lines: 13

Contained by: file nodes.property_extractor.spin_parameter.F90
Modules used: galacticus_nodes

function: spinname
Description: Return the name of the spin property.
Code lines: 9
Contained by: file nodes.property_extractor.spin_parameter.F90

function: spintype
Description: Return the type of the spin parameter property.
Code lines: 9
Contained by: file nodes.property_extractor.spin_parameter.F90
Modules used: output_analyses_options

function: spinunitsinsi
Description: Return the units of the spin property in the SI system.
Code lines: 8
Contained by: file nodes.property_extractor.spin_parameter.F90

file: nodes.property_extractor.tree_indices.F90
Description: Contains a module which implements an ISM mass output analysis property extractor class.
Code lines: 100

function: indicestreeconstructorparameters
Description: Constructor for the indicesTree node property extractor class which takes a parameter set as input.
Code lines: 10
Contained by: file nodes.property_extractor.tree_indices.F90
Modules used: input_parameters

function: indicestreedescription
Description: Return a description of the indicesTree property.
Code lines: 9
Contained by: file nodes.property_extractor.tree_indices.F90

function: indicestreeextract
Description: Implement a indicesTree node property extractor.
Code lines: 12
Contained by: file nodes.property_extractor.tree_indices.F90

function: indicestreename
Description: Return the name of the indicesTree property.
Code lines: 9
Contained by: file nodes.property_extractor.tree_indices.F90

function: indicestreetype
Description: Return the type of the stellar mass property.
Code lines: 9
Contained by: file nodes.property_extractor.tree_indices.F90

Modules used: output_analyses_options

interface: nodepropertyextractorindicestree
Description: Constructors for the "indicesTree" output analysis class.
Code lines: 3
Contained by: file nodes.property_extractor.tree_indices.F90

file: nodes.property_extractor.tree_weight.F90
Description: Contains a module which implements a merger tree weight property extractor class.
Code lines: 109

interface: nodepropertyextractortreeweight
Description: Constructors for the "treeWeight" output analysis class.
Code lines: 3
Contained by: file nodes.property_extractor.tree_weight.F90

function: treeweightconstructorparameters
Description: Constructor for the treeWeight property extractor class which takes a parameter set as input.
Code lines: 10
Contained by: file nodes.property_extractor.tree_weight.F90
Modules used: input_parameters

function: treeweightdescription
Description: Return a description of the treeWeight property.
Code lines: 9
Contained by: file nodes.property_extractor.tree_weight.F90

function: treeweightextract
Description: Implement a last isolated redshift output analysis.
Code lines: 10
Contained by: file nodes.property_extractor.tree_weight.F90

function: treeweightname
Description: Return the name of the last isolated redshift property.
Code lines: 9
Contained by: file nodes.property_extractor.tree_weight.F90

function: treeweighttype
Description: Return the type of the last isolated redshift property.
Code lines: 9
Contained by: file nodes.property_extractor.tree_weight.F90
Modules used: output_analyses_options

function: treeweightunitsinsi
Description: Return the units of the last isolated redshift property in the SI system.
Code lines: 9
Contained by: file nodes.property_extractor.tree_weight.F90
Modules used: numerical_constants_astronomical

file: `nodes.property_extractor.tuple.F90`
Code lines: 111

type: `nodepropertyextractortuple`
Description: A tuple property extractor.
Code lines: 47
Contained by: file `nodes.property_extractor.tuple.F90`

file: `nodes.property_extractor.velocity_dispersion.F90`
Description: Contains a module which implements a property extractor class for the velocity dispersion at a set of radii.
Code lines: 543
Modules used: `dark_matter_halo_scales` `galactic_structure_radii_definitions`

interface: `nodepropertyextractorvelocitydispersion`
Description: Constructors for the "velocityDispersion" output analysis class.
Code lines: 4
Contained by: file `nodes.property_extractor.velocity_dispersion.F90`

function: `velocitydispersionconstructorinternal`
Description: Internal constructor for the `velocityDispersion` property extractor class.
Code lines: 19
Contained by: file `nodes.property_extractor.velocity_dispersion.F90`
Modules used: `galactic_structure_radii_definitions`

function: `velocitydispersionconstructorparameters`
Description: Constructor for the `velocityDispersion` property extractor class which takes a parameter set as input.
Code lines: 31
Contained by: file `nodes.property_extractor.velocity_dispersion.F90`
Modules used: `input_parameters`

function: `velocitydispersiondensityintegrand`
Description: Integrand function used for computing line-of-sight velocity dispersions.
Code lines: 12
Contained by: file `nodes.property_extractor.velocity_dispersion.F90`
Modules used: `galactic_structure_densities`

function: `velocitydispersiondescriptions`
Description: Return descriptions of the `velocityDispersion` property.
Code lines: 15
Contained by: file `nodes.property_extractor.velocity_dispersion.F90`

subroutine: `velocitydispersiondestructor`
Description: Destructor for the `velocityDispersion` property extractor class.
Code lines: 7
Contained by: file `nodes.property_extractor.velocity_dispersion.F90`

function: `velocitydispersionelementcount`

Description: Return the number of elements in the `velocityDispersion` property extractors.
Code lines: 9
Contained by: file `nodes.property_extractor.velocity_dispersion.F90`

function: `velocitydispersionextract`

Description: Implement a `velocityDispersion` property extractor.
Code lines: 133
Contained by: file `nodes.property_extractor.velocity_dispersion.F90`
Modules used: `fgsl`, `galactic_structure_enclosed_masses`, `galactic_structure_options`, `galactic_structure_radii_definitions`, `galactic_structure_velocity_dispersions`, `galacticus_nodes`, `numerical_integration`

function: `velocitydispersionlambdarintegrand1`

Description: Integrand function used for integrating the λ_R statistic of Cappellari et al. [2007]. In this case we want to evaluate

$$\int_0^r 2\pi r' \Sigma(r') \sqrt{\sigma^2(r') + V^2(r')} \mathrm{d}r', \tag{18.24}$$

where $\Sigma(r)$ is the projected surface density (in mass or light) of the galaxy at radius r, $\sigma^2(r)$ is the measured velocity dispersion and $V(r)$ the measured rotation speed. Assuming that the selected component is purely dispersion dominated with velocity dispersion $\sigma_\mathrm{s}(r)$, and that rotation is present in only the disk component with rotation curve $V_\mathrm{d}(r)$ then we can model the velocity distribution, $P(V)$, at r as the sum of a Gaussian of width $\sigma_\mathrm{s}(r)$ and normalized area $\Sigma_\mathrm{s}(r)$, and a delta function at $V_\mathrm{d}(r)$ with normalized area $\Sigma_\mathrm{d}(r)$. The measured rotation speed is then:

$$V(r) = \int_{-\infty}^{+\infty} P(V) V \mathrm{d}V \Big/ \int_{-\infty}^{+\infty} P(V) \mathrm{d}V = \frac{\Sigma_\mathrm{d}(r) V_\mathrm{d}(r)}{[\Sigma_\mathrm{d}(r) + \Sigma_\mathrm{s}(r)]}, \tag{18.25}$$

and the measured velocity dispersion is:

$$\sigma^2(r) = \int_{-\infty}^{+\infty} P(V) [V - V(r)]^2 \mathrm{d}V \Big/ \int_{-\infty}^{+\infty} P(V) \mathrm{d}V = \frac{\Sigma_\mathrm{s}(r) [\sigma_\mathrm{s}^2(r)] + \Sigma_\mathrm{d}(r) [V_\mathrm{d}(r) - V(r)]^2}{[\Sigma_\mathrm{d}(r) + \Sigma_\mathrm{s}(r)]}. \tag{18.26}$$

Code lines: 54
Contained by: file `nodes.property_extractor.velocity_dispersion.F90`
Modules used: `fgsl`, `galactic_structure_options`, `galactic_structure_rotation_curves`, `galactic_structure_surface_densities`, `numerical_constants_math`, `numerical_integration`

function: `velocitydispersionlambdarintegrand2`

Description: Integrand function used for integrating the $\lambda_{\rm R}$ statistic of Cappellari et al. [2007]. In this case we want to evaluate

$$\int_0^r 2\pi r' \Sigma(r') V(r') \mathrm{d}r', \tag{18.27}$$

where $\Sigma(r)$ is the projected surface density (in mass or light) of the galaxy at radius r, and $V(r)$ the measured rotation speed. Assuming that the selected component is purely dispersion dominated with velocity dispersion $\sigma_{\rm s}(r)$, and that rotation is present in only the disk component with rotation curve $V_{\rm d}(r)$ then we can model the velocity distribution, $P(V)$, at r as the sum of a Gaussian of width $\sigma_{\rm s}(r)$ and normalized area $\Sigma_{\rm s}(r)$, and a delta function at $V_{\rm d}(r)$ with normalized area $\Sigma_{\rm d}(r)$. The measured rotation speed is then:

$$V(r) = \int_{-\infty}^{+\infty} P(V) V \mathrm{d}V \Big/ \int_{-\infty}^{+\infty} P(V) \mathrm{d}V = \frac{\Sigma_{\rm d}(r) V_{\rm d}(r)}{[\Sigma_{\rm d}(r) + \Sigma_{\rm s}(r)]}. \tag{18.28}$$

Code lines: 31

Contained by: file `nodes.property_extractor.velocity_dispersion.F90`

Modules used: galactic_structure_options galactic_structure_rotation_curves galactic_structure_surface_densities numerical_constants_math

function: velocitydispersionlineofsightvelocitydispersionintegrand
Description: Compute the line-of-sight velocity dispersion at the given radius.
Code lines: 20
Contained by: file nodes.property_extractor.velocity_dispersion.F90
Modules used: fgsl numerical_integration

function: velocitydispersionnames
Description: Return the names of the velocityDispersion properties.
Code lines: 15
Contained by: file nodes.property_extractor.velocity_dispersion.F90

function: velocitydispersionsolidangleincylinder
Description: Computes the solid angle of a spherical shelll of given radius that lies within a cylinder of radius radiusImpact.
Code lines: 16
Contained by: file nodes.property_extractor.velocity_dispersion.F90
Modules used: numerical_constants_math

function: velocitydispersionsurfacedensityintegrand
Description: Integrand function used for integrating line-of-sight surface density dispersion over area.
Code lines: 12
Contained by: file nodes.property_extractor.velocity_dispersion.F90
Modules used: galactic_structure_densities

function: velocitydispersiontype
Description: Return the type of the velocityDispersion properties.
Code lines: 9
Contained by: file nodes.property_extractor.velocity_dispersion.F90
Modules used: output_analyses_options

function: velocitydispersionunitsinsi
Description: Return the units of the velocityDispersion properties in the SI system.
Code lines: 17
Contained by: file nodes.property_extractor.velocity_dispersion.F90
Modules used: numerical_constants_astronomical numerical_constants_prefixes

function: velocitydispersionvelocitydensityintegrand

Description: Integrand function used for computing line-of-sight velocity dispersions. Specifically, we wish to evaluate the integral:

$$\int_{r_\mathrm{i}}^{r_\mathrm{o}} \sigma^2(r)\rho(r)\frac{r}{\sqrt{r^2-r_\mathrm{i}^2}}\mathrm{d}r, \tag{18.29}$$

where r_i is the impact parameter, r_o is an outer radius at which we assume $\rho(r_\mathrm{o})\sigma^2(r_\mathrm{o}) = 0$ (i.e. it is the radius at which we begin integrating the Jeans equation), $\rho(r)$ is density, and $\sigma(r)$ is the velocity dispersion at radius r. Assuming spherical symmetry and isotropic velocity dispersion, the Jeans equation tells us

$$\rho(r)\sigma^2(r) = \int_r^{r_\mathrm{o}} \frac{\mathrm{G}M(<r')}{r'^2}\rho(r')\mathrm{d}r', \tag{18.30}$$

where G is the gravitational constant, and $M(<r)$ is the total mass contained within radius r. Equation (18.29) can then be simplified using integration by parts to give:

$$\left[\sigma^2(r)\rho(r)\sqrt{r^2-r_\mathrm{i}^2}\right]_{r_\mathrm{i}}^{r_\mathrm{o}} + \int_{r_\mathrm{i}}^{r_\mathrm{o}} \frac{\mathrm{d}}{\mathrm{d}r}\left[\sigma^2(r)\rho(r)\right]\sqrt{r^2-r_\mathrm{i}^2}\mathrm{d}r. \tag{18.31}$$

The first term is zero at both limits (due to the constraint $\rho(r_\mathrm{o})\sigma^2(r_\mathrm{o}) = 0$ at r_o and due to $sqrtr^2 - r_\mathrm{i}^2 = 0$ at r_i), and the second term can be simplified using eqn. (18.30) to give

$$\int_{r_\mathrm{i}}^{r_\mathrm{o}} \frac{\mathrm{G}M(<r)}{r^2}\rho(r)\sqrt{r^2-r_\mathrm{i}^2}\mathrm{d}r. \tag{18.32}$$

Code lines: 38
Contained by: file nodes.property_extractor.velocity_dispersion.F90
Modules used: galactic_structure_densities, galactic_structure_enclosed_masses, galactic_structure_options, numerical_constants_physical

function: `velocitydispersionvelocitysurfacedensityintegrand`

Description: Integrand function used for integrating line-of-sight velocity dispersion over surface density.
Code lines: 13
Contained by: file nodes.property_extractor.velocity_dispersion.F90
Modules used: galactic_structure_densities galactic_structure_velocity_dispersions

file: nodes.property_extractor.velocity_maximum.F90
Description: Contains a module which implements a cooling rate property extractor class.
Code lines: 138
Modules used: dark_matter_profiles_dmo

interface: nodepropertyextractorvelocitymaximum
Description: Constructors for the "velocityMaximum" output analysis class.
Code lines: 4
Contained by: file nodes.property_extractor.velocity_maximum.F90

function: velocitymaximumconstructorinternal
Description: Internal constructor for the velocityMaximum property extractor class.
Code lines: 8
Contained by: file nodes.property_extractor.velocity_maximum.F90

function: velocitymaximumconstructorparameters
Description: Constructor for the velocityMaximum property extractor class which takes a parameter set as input.
Code lines: 13
Contained by: file nodes.property_extractor.velocity_maximum.F90
Modules used: input_parameters

function: velocitymaximumdescription
Description: Return a description of the velocityMaximum property.
Code lines: 9
Contained by: file nodes.property_extractor.velocity_maximum.F90

subroutine: velocitymaximumdestructor
Description: Destructor for the velocityMaximum property extractor class.
Code lines: 7
Contained by: file nodes.property_extractor.velocity_maximum.F90

function: velocitymaximumextract
Description: Implement a last isolated redshift output analysis.
Code lines: 11
Contained by: file nodes.property_extractor.velocity_maximum.F90
Modules used: galacticus_nodes

function: velocitymaximumname
Description: Return the name of the last isolated redshift property.
Code lines: 9
Contained by: file nodes.property_extractor.velocity_maximum.F90

function: `velocitymaximumtype`
Description: Return the type of the last isolated redshift property.
Code lines: 9
Contained by: file `nodes.property_extractor.velocity_maximum.F90`
Modules used: `output_analyses_options`

function: `velocitymaximumunitsinsi`
Description: Return the units of the last isolated redshift property in the SI system.
Code lines: 9
Contained by: file `nodes.property_extractor.velocity_maximum.F90`
Modules used: `numerical_constants_prefixes`

file: `nodes.property_extractor.virial_properties.F90`
Code lines: 157
Modules used: `dark_matter_halo_scales`

interface: `nodepropertyextractorvirialproperties`
Description: Constructors for the "virialProperties" output extractor class.
Code lines: 4
Contained by: file `nodes.property_extractor.virial_properties.F90`

function: `virialpropertiesconstructorinternal`
Description: Internal constructor for the "virialProperties" output extractor property extractor class.
Code lines: 8
Contained by: file `nodes.property_extractor.virial_properties.F90`

function: `virialpropertiesconstructorparameters`
Description: Constructor for the "virialProperties" property extractor class which takes a parameter set as input.
Code lines: 13
Contained by: file `nodes.property_extractor.virial_properties.F90`
Modules used: `input_parameters`

function: `virialpropertiesdescriptions`
Description: Return the descriptions of the `virialProperies` properties.
Code lines: 11
Contained by: file `nodes.property_extractor.virial_properties.F90`

subroutine: `virialpropertiesdestructor`
Description: Destructor for the `virialProperties` property extractor class.
Code lines: 7
Contained by: file `nodes.property_extractor.virial_properties.F90`

function: `virialpropertieselementcount`
Description: Return the number of elements in the `virialProperies` property extractors.
Code lines: 9
Contained by: file `nodes.property_extractor.virial_properties.F90`

function: `virialpropertiesextract`
Description: Implement a virialProperties output extractor.
Code lines: 13
Contained by: file `nodes.property_extractor.virial_properties.F90`

function: `virialpropertiesnames`
Description: Return the names of the `virialProperies` properties.
Code lines: 11
Contained by: file `nodes.property_extractor.virial_properties.F90`

function: `virialpropertiestype`
Description: Return the type of the virialProperties property.
Code lines: 9
Contained by: file `nodes.property_extractor.virial_properties.F90`
Modules used: `output_analyses_options`

function: `virialpropertiesunitsinsi`
Description: Return the units of the `virialProperies` properties in the SI system.
Code lines: 13
Contained by: file `nodes.property_extractor.virial_properties.F90`
Modules used: `numerical_constants_astronomical` `numerical_constants_prefixes`

file: `numerical.FFTlog.F90`
Description: Contains a module which wraps the FFTLog functions.
Code lines: 83

module: `fftlogs`
Description: Wraps the FFTLog functions.
Code lines: 61
Contained by: file `numerical.FFTlog.F90`
Used by: subroutine `correlationfunctionfinalizeanalysis` subroutine `halo_model_projected_correlation`

subroutine: `fftlog`
Description: Perform a discrete FFT on logarithmically spaced data.
Code lines: 38
Contained by: module `fftlogs`
Modules used: `galacticus_error` `numerical_constants_math`

file: `numerical.ODE_solver.F90`
Description: Contains a module which provides an interface to the GNU Scientific Library ODEIV differential equation solvers.
Code lines: 153

module: `ode_solver`
Description: Contains an interface to the GNU Scientific Library ODEIV differential equation solvers.
Code lines: 131
Contained by: file `numerical.ODE_solver.F90`

Modules used:	`iso_c_binding`	ode_solver_error_codes
Used by:	function matterdarkenergyexpansionfactorchange	function matterlambdaconstructorinternal
	subroutine matterlambdamakeexpansionfactortable	function zhao2009time
	subroutine simpleretabulate	program test_ode_solver

subroutine: `ode_solve`

Description:	Interface to the GNU Scientific Library ODEIV differential equation solvers.	
Code lines:	74	
Contained by:	module ode_solver	
Modules used:	`fgsl`	galacticus_error
	`iso_c_binding`	

subroutine: `ode_solver_free`

Description:	Free up workspace allocated to ODE solving.
Code lines:	14
Contained by:	module ode_solver
Modules used:	`fgsl`

function: `odeswrapper`

Description:	Wrapper function used for GNU Scientific Library (GSL) ODE functions.
Code lines:	13
Contained by:	module ode_solver
Modules used:	`iso_c_binding`

file: `numerical.ODE_solver.ODEIV2.F90`

Description:	Contains a module which provides an interface to the GNU Scientific Library ODEIV2 differential equation solvers.
Code lines:	380

module: `odeiv2_solver`

Description:	Contains an interface to the GNU Scientific Library ODEIV2 differential equation solvers.	
Code lines:	358	
Contained by:	file numerical.ODE_solver.ODEIV2.F90	
Modules used:	fodeiv2	`iso_c_binding`
	ode_solver_error_codes	
Used by:	subroutine gnedin2000tabulate	subroutine standarddestructor
	subroutine standardevolve	function intergalacticbackgroundinternalupdate
	subroutine perturbation_dynamics_solver	program test_ode_solver
	function intergalacticmediumstateevolveupdate	

function: `jacobianwrapperiv2`

Description:	Wrapper function used for GSL ODEIV2 Jacobian functions.
Code lines:	13
Contained by:	module odeiv2_solver
Modules used:	`iso_c_binding`

subroutine: `odeiv2_solve`
Description: Interface to the GNU Scientific Library ODEIV2 differential equation solvers.
Code lines: 252
Contained by: module `odeiv2_solver`
Modules used: `fgsl`, `galacticus_error`, `iso_varying_string`, `numerical_integration2`, `string_handling`

subroutine: `integrandswrapper`
Description: Wrapper function which calls the integrands functions.
Code lines: 17
Contained by: subroutine `odeiv2_solve`

subroutine: `latentintegrator`
Description: Wrapper function which performs integration of latent variables.
Code lines: 25
Contained by: subroutine `odeiv2_solve`
Modules used: `galacticus_display`, `galacticus_error`

subroutine: `odeiv2_solver_free`
Description: Free up workspace allocated to ODE solving.
Code lines: 9
Contained by: module `odeiv2_solver`

type: `odeiv2odeslist`
Description: Type used to maintain a list of ODEs when ODE solving is performed recursively.
Code lines: 7
Contained by: module `odeiv2_solver`

function: `odeswrapperiv2`
Description: Wrapper function used for GSL ODEIV2 functions.
Code lines: 13
Contained by: module `odeiv2_solver`
Modules used: `iso_c_binding`

file: `numerical.ODE_solver.ODEIV2.wrapper.F90`
Code lines: 612

module: `fodeiv2`
Code lines: 592
Contained by: file `numerical.ODE_solver.ODEIV2.wrapper.F90`
Modules used: `fgsl`, `iso_c_binding`
Used by: subroutine `gnedin2000tabulate`, file `merger_trees.node_-evolver.standard.F90`, function `standardconstructorinternal`, subroutine `standarderrorhandler`, function `standardodes`, module `odeiv2_solver`, function `intergalacticbackgroundinternalupdate`, subroutine `perturbation_dynamics_solver`

program test_ode_solver

function intergalacticmediumstateevolveupdate

type: fodeiv2_control
Code lines: 2
Contained by: module fodeiv2

subroutine: fodeiv2_control_free
Code lines: 3
Contained by: module fodeiv2

function: fodeiv2_control_hadjust
Code lines: 7
Contained by: module fodeiv2

function: fodeiv2_control_init
Code lines: 5
Contained by: module fodeiv2

function: fodeiv2_control_name
Code lines: 7
Contained by: module fodeiv2

function: fodeiv2_control_scaled2_new
Code lines: 6
Contained by: module fodeiv2

function: fodeiv2_control_scaled_new
Code lines: 6
Contained by: module fodeiv2

function: fodeiv2_control_standard_new
Code lines: 4
Contained by: module fodeiv2

function: fodeiv2_control_status
Code lines: 5
Contained by: module fodeiv2

function: fodeiv2_control_y_new
Code lines: 4
Contained by: module fodeiv2

function: fodeiv2_control_yp_new
Code lines: 4
Contained by: module fodeiv2

type: fodeiv2_driver
Code lines: 2
Contained by: module fodeiv2

function: `fodeiv2_driver_alloc_scaled_new`
Code lines: 10
Contained by: module `fodeiv2`

function: `fodeiv2_driver_alloc_standard_new`
Code lines: 8
Contained by: module `fodeiv2`

function: `fodeiv2_driver_alloc_y_new`
Code lines: 8
Contained by: module `fodeiv2`

function: `fodeiv2_driver_alloc_yp_new`
Code lines: 8
Contained by: module `fodeiv2`

function: `fodeiv2_driver_apply`
Code lines: 11
Contained by: module `fodeiv2`

function: `fodeiv2_driver_apply_fixed_step`
Code lines: 9
Contained by: module `fodeiv2`

subroutine: `fodeiv2_driver_errors`
Code lines: 4
Contained by: module `fodeiv2`

subroutine: `fodeiv2_driver_free`
Code lines: 3
Contained by: module `fodeiv2`

function: `fodeiv2_driver_h`
Code lines: 4
Contained by: module `fodeiv2`

subroutine: `fodeiv2_driver_msbdfactive_context`
Code lines: 6
Contained by: module `fodeiv2`

subroutine: `fodeiv2_driver_msbdfactive_state`
Code lines: 5
Contained by: module `fodeiv2`

function: `fodeiv2_driver_reset`
Code lines: 4
Contained by: module `fodeiv2`

function: `fodeiv2_driver_set_hmax`
Code lines: 5
Contained by: module `fodeiv2`

function: `fodeiv2_driver_set_hmin`
Code lines: 5
Contained by: module `fodeiv2`

function: `fodeiv2_driver_set_nmax`
Code lines: 5
Contained by: module `fodeiv2`

function: `fodeiv2_driver_status`
Code lines: 5
Contained by: module `fodeiv2`

type: `fodeiv2_evolve`
Code lines: 2
Contained by: module `fodeiv2`

function: `fodeiv2_evolve_alloc`
Code lines: 4
Contained by: module `fodeiv2`

function: `fodeiv2_evolve_apply`
Code lines: 10
Contained by: module `fodeiv2`

subroutine: `fodeiv2_evolve_free`
Code lines: 3
Contained by: module `fodeiv2`

function: `fodeiv2_evolve_reset`
Code lines: 4
Contained by: module `fodeiv2`

function: `fodeiv2_evolve_status`
Code lines: 5
Contained by: module `fodeiv2`

type: `fodeiv2_step`
Code lines: 2
Contained by: module `fodeiv2`

function: `fodeiv2_step_alloc`
Code lines: 12
Contained by: module `fodeiv2`

function: `fodeiv2_step_apply`

Code lines: 7
Contained by: module fodeiv2

subroutine: `fodeiv2_step_free`
Code lines: 3
Contained by: module fodeiv2

function: `fodeiv2_step_name`
Code lines: 7
Contained by: module fodeiv2

function: `fodeiv2_step_order`
Code lines: 4
Contained by: module fodeiv2

function: `fodeiv2_step_reset`
Code lines: 4
Contained by: module fodeiv2

function: `fodeiv2_step_status`
Code lines: 5
Contained by: module fodeiv2

type: `fodeiv2_step_type`
Code lines: 3
Contained by: module fodeiv2

type: `fodeiv2_system`
Code lines: 3
Contained by: module fodeiv2

subroutine: `fodeiv2_system_free`
Code lines: 3
Contained by: module fodeiv2

function: `fodeiv2_system_init`
Code lines: 34
Contained by: module fodeiv2

function: `fodeiv2_system_status`
Code lines: 5
Contained by: module fodeiv2

file: `numerical.ODE_solver.error_codes.F90`
Description: Contains a module which defines internal error codes for the Galacticus ODE solver.
Code lines: 34

module: `ode_solver_error_codes`
Description: Defines internal error codes for the Galacticus ODE solver.

Code lines: 12
Contained by: file numerical.ODE_solver.error_codes.F90
Modules used: fgsl
Used by:

subroutine standardevolve	function standardodes
module ode_solver	module odeiv2_solver
function intergalacticbackgroundinternalodes	function intergalacticmediumstateevolveodes

file: numerical.comparison.F90
Description: Contains a module which implements comparisons of values.
Code lines: 120

module: numerical_comparison
Description: Implements comparisons of values.
Code lines: 98
Contained by: file numerical.comparison.F90
Used by:

function matterlambdaconstructorinternal	function dark_matter_profile_mass_definition
function ludlow2016radius	function concentrationradius
function burkertradialmoment	function radialmoment
function einastoradialmoment	function nfwradialmoment
function isothermalradialmoment	function hivshalomassrelationpadmanabhan2017constructorin
function localgroupmassfunctionconstructorinterna	function blackholebulgerelationconstructorinternal
function colordistributionsdssconstructorinternal	function concentrationdistributioncdmcococonstructorinter
function concentrationvshalomasscdmludlow2016cons	function galaxysizeconstructorinternal
function spindistributionbett2007constructorinterna	function stellarvshalomassrelationleauthaud2012constructo
function squareisinlightcone	function squareposition
function miyamotonagaiconstructorinternal	function exponentialdiskconstructorinternal
function hernquistconstructorinternal	function hernquistdensityradialmoment
function nfwconstructorinternal	function sersicconstructorinternal
function sersicdensityradialmoment	function betaprofileconstructorinternal
function betaprofiledensityradialmoment	function betaprofilepotential
function parkinsoncolehellyprobabilitybound	function buildconstructorinternal
subroutine cole2000build	function readconstruct
subroutine sussingasciiopen	subroutine sussinghdf5open
subroutine galacticusopen	function standardisaccurate
function augmentbuildtreefromnode	subroutine conditionalmfoperate
subroutine outputrootmassesoperate	subroutine particulateoperate
subroutine pruneclonesoperate	subroutine regridtimesoperate

function adaptivecompositetrapezoidalevaluate1d	function compositegausskronrod1devaluate
function compositetrapezoidalevaluate1d	function multivectorizedcompositegausskronrod1devaluate
function multivectorizedcompositetrapezoidal1devaluate	function vectorizedcompositegausskronrod1devaluate
function vectorizedcompositetrapezoidalevaluate1d	function search_array_for_closest
function stellar_luminosities_index_-from_properties	function listindex
subroutine cole2000get	subroutine covington2008get
function zhanghuig2integrated	subroutine takahashi2011lensingdistributionconstruct
function cosmicemuconstructorinternal	function cosmicemuvalue
subroutine filereadfile	function bryannorman1998constructorinternal
function kitayamasuto1996constructorinternal	program test_comparison
function intergalacticmediumstateevolveconstructorinternal	function array_is_uniform
subroutine assert_double_1d_array	subroutine assert_double_2d_array
subroutine assert_double_3d_array	subroutine assert_double_4d_array
subroutine assert_double_5d_array	subroutine assert_double_complex_1d_-array
subroutine assert_double_scalar	subroutine assert_real_1d_array
subroutine assert_real_scalar	

interface: `values_agree`
Code lines: 3
Contained by: module numerical_comparison

function: `values_agree_double`
Description: Returns true if `value1` and `value2` agree to within `absTol` in absolute terms, or `relTol` in relative terms.
Code lines: 24
Contained by: module numerical_comparison

function: `values_agree_real`
Description: Returns true if `value1` and `value2` agree to within `absTol` in absolute terms, or `relTol` in relative terms.
Code lines: 24
Contained by: module numerical_comparison

interface: `values_differ`
Code lines: 3
Contained by: module numerical_comparison

function: `values_differ_double`
Description: Returns true if `value1` and `value2` differ by more than `absTol` in absolute terms, or `relTol` in relative terms.

Code lines: 12
Contained by: module numerical_comparison

function: values_differ_real
Description: Returns true if value1 and value2 differ by more than absTol in absolute terms, or relTol in relative terms.
Code lines: 12
Contained by: module numerical_comparison

file: numerical.constants.astronomical.F90
Description: Contains a module of useful astronomical constants.
Code lines: 80

module: numerical_constants_astronomical
Description: Contains various useful astronomical constants.
Code lines: 58
Contained by: file numerical.constants.astronomical.F90
Modules used: numerical_constants_atomic numerical_constants_units
Used by:

function bondi_hoyle_lyttleton_accretion_rate	function coldmodechemicalmasses
function coldmodecoldmodefraction	function simplechemicalmasses
function shakurasunyaevpowerjet	subroutine hopkins2007buildfile
function filespectrum	function standardgrowthrate
function black_hole_eddington_accretion_rate	function chemicals_mass_to_density_conversion
function simpletime	function matterlambdacomovingdistanceintegrand
function simplehubbleconstant	function simpleomegaradiation
function virialdensitycontrastdefinitiondynamicaltimescale	function virialdensitycontrastdefinitionvirialtemperature
function integrandtimefreefall	function twobodyrelaxationspecificenergy
function burkertfreefallradius	function burkertfreefallradiusincreaserate
function einastofreefallradius	function einastofreefallradiusincreaserate
function nfwfreefallradius	function nfwfreefallradiusincreaserate
function isothermalfreefallradius	function isothermalfreefallradiusincreaserate
function efstathiou1982estimator	subroutine efstathiou1982timescale
function efstathiou1982tidalestimator	subroutine stabletimescale
subroutine galacticus_meta_evolver_profiler_output	subroutine meta_tree_timing_output
function hivshalomassrelationpadmanabhan2017constructorinternal	function localgroupmassfunctionconstructorinternal
subroutine localgroupmassfunctionfinalize	function blackholebulgerelationconstructorinternal

function colordistributionsdssconstructorinternal
function concentrationvshalomasscdmludlow2016constructorinternal
subroutine correlationfunctionfinalize
function galaxysizessdssconstructorinternal
function luminosityfunctionconstructorinternal
function luminosityfunctionhalphaconstructorinternal
function massfunctionhiconstructorinternal
function massfunctionstellarconstructorinternal
function massmetallicityandrews2013constructorinternal
function massmetallicityblanc2017constructorinternal
function obreschkow2009ratio
function morphologicalfractiongamamoffett2016constructorinternal
function himassoperate
function stellarvshalomassrelationleauthaud2012constructorinternal
subroutine galacticus_output_halo_model_initialize
function squareconstructorparameters
function gunawardhana2013sdssdistancemaximum
subroutine martin2010alfalfarandomsinitialize
function rampressureaccelerationtimescale
function enzohydrostatictemperature
subroutine interface_camb_transfer_function
subroutine interface_fsps_ssps_tabulate
subroutine localgroupdbupdate
function gnedin2000odes
function massfilteringodes
function intergalacticmediumstateelectronscatteringintegrand
function recfastconstructorinternal
function instantreionizationelectronfraction
function instantreionizationneutralheliumfraction
function instantreionizationneutralhydrogenfraction
function instantreionizationsinglyionizedheliumfraction
function simpleelectronfraction
function simpleneutralheliumfraction
function simpleneutralhydrogenfraction
function simplesinglyionizedheliumfraction
function readconstructorinternal
subroutine sussingasciiload
subroutine sussingasciiopen
function sussingcubelength
function sussinginsubvolume1d
subroutine sussingtreeindicesread
function sussingtreeweight
subroutine sussinghdf5load
subroutine sussinghdf5open
function galacticuscubelength
subroutine galacticusforestindicesread
subroutine galacticusimport
subroutine galacticussubhalotrace
function galacticustreeweight
subroutine historywrite
subroutine recordevolutionoutput
subroutine conditionalmffinalize
subroutine exportoperate
subroutine massaccretionhistoryoperate
subroutine merger_tree_structure_output
subroutine standardmakegroup
subroutine standardoutputgroupcreate
subroutine merger_trees_render_dump

function gadgetbinaryimport
function integrandcomptiony
function rateinfallcoldmodeunitsinsi
function ratecoolingunitsinsi
function densitycontrastsunitsinsi

function radiushalfmassunitsinsi
function lightconeconstructorinternal
function lmnstyemssnlineextract
function lmnstystllrcf2000unitsinsi
function massismunitsinsi
function masshalounitsinsi
function massstellarunitsinsi
function projecteddensityunitsinsi
function rotationcurveunitsinsi

function treeweightunitsinsi
function virialpropertiesunitsinsi
subroutine kepler_orbits_output_names
subroutine node_component_black_hole_-noncentral_rate_compute
subroutine node_component_black_hole_-standard_mass_accretion_rate
subroutine node_component_black_hole_-standard_rate_compute
subroutine node_component_disk_-standard_rate_compute
subroutine node_component_hot_halo_-cold_mode_formation
subroutine node_component_hot_halo_-cold_mode_rate_compute
subroutine node_component_hot_halo_-standard_outflow_return
subroutine node_component_satellite_-orbiting_rate_compute
subroutine node_component_spheroid_-standard_rate_compute
function intergalacticbackgroundinternalflux
function simpleratemassloss
function chandrasekhar1943acceleration
function zentner2005masslossrate
function blitz2006constructorinternal
function dynamicaltimetimescale
function standardenergyinputcumulative

function gadgethdf5import
function integrandluminosityxray
function radiuscoolingunitsinsi
function densityprofileunitsinsi
function radiihalflightpropertiesunitsinsi
function masshostunitsinsi
function lightconeextract
function luminositystellarunitsinsi
function lmnstystllrchrltfll2000extract
function massblackholeunitsinsi
function massprofileunitsinsi
function massstellarspheroidunitsinsi
function halfmassradiusunitsinsi
function satelliteorbitalextremaunitsinsi
function velocitydispersionunitsinsi
module abundances_structure
module galacticus_nodes
subroutine node_component_black_hole_-simple_output_names
subroutine node_component_black_hole_-standard_output_names
subroutine node_component_dark_matter_-profile_scale_tree_output
subroutine node_component_dynamics_-statistics_bars_output
subroutine node_component_hot_halo_-cold_mode_outflow_return
subroutine node_component_hot_halo_-standard_formation
subroutine node_component_hot_halo_-standard_rate_compute
subroutine node_component_satellite_-orbiting_scale_set
module tensors

function intergalacticbackgroundinternalodes
function simpleratemassloss
function gnedin1999heatingrate
function zentner2005tidalradiussolver
function dynamicaltimetimescale
function standardconstructorparameters
function hegerwoosley2002constructorinternal

file stellar_-astrophysics.winds.Leitherer1992.F90	subroutine stellar_population_-luminosity_tabulate
function filter_luminosity_integrand_ab	function wittgordon2003constructorinternal
subroutine filetabulation	function filewavelengthinterval
subroutine filewavelengths	function standardinterpolate
function standardrecycledfractioninstantaneous	function standardyieldinstantaneous
function cambconstructorinternal	subroutine catalogprojectedcorrelationfunctionperform
subroutine halomassfunctionperform	subroutine halomodelgenerateperform
subroutine massfunctioncovarianceperform	function mergertreefilebuilderconstructorparameters
subroutine powerspectraperform	program test_cooling_functions
program test_dark_matter_profiles_-heated	program test_stellar_populations
program test_stellar_populations_-luminosities	function ideal_gas_sound_speed
function simpleratemassloss	function simpleratemassloss
function intergalacticmediumstateevolveconstructorinternal	function intergalacticmediumstateevolveodes
function intergalacticmediumstateevolveupdate	subroutine iratereadhalos
subroutine iratewritehalos	

file: numerical.constants.atomic.F90
Description: Contains a module of useful atomic constants.
Code lines: 43

module: numerical_constants_atomic
Description: Contains various useful atomic constants.
Code lines: 21
Contained by: file numerical.constants.atomic.F90
Modules used: fgsl numerical_constants_physical numerical_constants_units
Used by:

function coldmodecoldmodefraction	function metallicity12lognhoperate
function enzohydrostatictemperature	function massfilteringodes
function integrandcomptiony	function integrandluminosityxray
function lmnstyemssnlineextract	function lmnstystllrchrltfll2000extract
module numerical_constants_-astronomical	module chemical_structures
subroutine node_component_hot_halo_-cold_mode_outflow_return	subroutine node_component_hot_halo_-standard_outflow_return
function intergalacticbackgroundinternalflux	function intergalacticbackgroundinternalodes
function inoue2014multiplier	function lycsuppressmultiplier
function madau1995multiplier	function meiksin2006multiplier

program test_inoue2014

function intergalacticmediumstateevolveconstructorinternal

function intergalacticmediumstateevolveodes

file: numerical.constants.boolean.F90
Description: Contains a module of useful Boolean constants.
Code lines: 30

module: numerical_constants_boolean
Description: Contains various useful Boolean constants.
Code lines: 8
Contained by: file numerical.constants.boolean.F90
Used by:
- function readconstructorinternal
- subroutine readphasespacepositionrealize
- function sussingcubelength
- function sussingpositionsareperiodic
- function sussingtreesareselfcontained
- function sussingtreeshavesubhalos
- function sussingtreeweight
- function sussingvelocitiesincludehubbleflow
- function galacticuscubelength
- function galacticuspositionsareperiodic
- function galacticustreesareselfcontained
- function galacticustreeshavesubhalos
- function galacticustreeweight
- function galacticusvelocitiesincludehubbleflow

file: numerical.constants.math.F90
Description: Contains a module of useful mathematical constants.
Code lines: 45

module: numerical_constants_math
Description: Contains various useful mathematical constants.
Code lines: 23
Contained by: file numerical.constants.math.F90
Modules used: fgsl, kind_numbers
Used by:
- function coldmodecoldmodefraction
- function campanelli2007velocity
- function black_hole_static_radius_spin
- function cole2000rate
- function whitefrenk1991rate
- function simpledensitycritical
- subroutine dark_matter_halo_-correa2015_fit_parameters
- function virialdensitycontrastdefinitionvirialradius
- function virialdensitycontrastdefinitionvirialradiusgradientlogmass
- subroutine nbodyerrorstabulate
- function lognormaldistribution
- function adiabaticgnedin2004derivativesolver
- function genericmassintegrand
- function genericenergynumerical
- function integrandfouriertransform
- function genericrotationnormalizationnumerical
- function rootcircularvelocitymaximum
- function rootdensity
- function diemerkravtsov2014concentrationmean
- function prada2011cmin

function prada2011inversesigmamin
function schneider2015referencecollapsemassroot
function ludlow2016radius
function dark_matter_profile_rotation_curve_gradient_task
function burkertdensityscalefree
function burkertkspace
function burkertprofileenergy
function radialmoment
function einastodensityscalefree
function einastofourierprofileintegrand
function einastoradiusenclosingdensityroot
function nfwdensityenclosedbyradiusscalefree
function nfwprofileenergy
function nfwradialvelocitydispersionscalefree
function isothermaldensity
function isothermalradiusenclosingdensity
function enclosed_density_root
function correlationfunctionloglikelihood
function disksizeinclntnintegrandx
function meanfunction1dloglikelihood
function volumefunction1dloglikelihood
function geometrymangleangularpower
subroutine caputi2011ukidssudsrandomsinitialize
subroutine martin2010alfalfarandomsinitialize
function fullskysolidangle
subroutine randompointswindowfunctions
function enzohydrostaticenclosedmassintegrand
function enzohydrostatictemperaturelogslope
function gnedin2000odes
function miyamotonagaiconstructorinternal
function schneider2015concentration
function dark_matter_profile_mass_definition
function concentrationradius
file dark_matter_profiles_DMO.Burkert.F90
function burkertfreefalltimescalefreeintegrand
function burkertpotential
function burkertradialmoment
subroutine burkertradiusenclosingdensitytabulate
subroutine einastoenergytablemake
function einastoradialmoment
function einastorotationnormalization
function nfwdensityscalefree
function nfwradialmoment
function heateddensity
function isothermalradialmoment
function truncatedexponentialenclosedmass
subroutine correlationfunctionfinalizeanalysis
function disksizeinclinationconstructorinternal
function disksizeinclntnroot
function scatterfunction1dloglikelihood
function squareconstructorinternal
function geometrymanglesolidangle
subroutine liwhite2009sdssrandomsinitialize
function fullskyangularpower
subroutine fullskywindowfunctions
subroutine halo_model_projected_correlation
function hothalomassdistributionrotationcurvegradient
subroutine gnedin2000conditionsinitialodes
subroutine gnedin2000tabulate
function miyamotonagaiconstructorparameters

function miyamotonagaimassenclosedbysphere
subroutine miyamotonagaimassenclosedtabulate
function exponentialdiskbesselfactorpotential
function exponentialdiskbesselfactorrotationcurve
function exponentialdiskbesselfactorrotationcurvegradient
function exponentialdiskconstructorinternal
function exponentialdiskconstructorparameters
function exponentialdiskmassenclosedbysphere
function sphericalmassenclosedbysphere
function hernquistconstructorinternal
function hernquistconstructorparameters
function hernquistdensityradialmoment
function hernquistmassenclosedbysphere
function nfwconstructorinternal
function nfwconstructorparameters
function sersicabelintegrand
function sersicconstructorinternal
function sersicdensityradialmoment
function sersicmassenclosedbysphere
subroutine sersictabulate
function betaprofileconstructorinternal
function betaprofilemassenclosedbysphere
function betaprofilepotential
function gaussian_distribution
function poisson_binomial_distribution
function poisson_binomial_distribution_jacobian
function erfapproximatequad
function error_function_complementary_complex
function exponential_integral_double_complex
subroutine parkinsoncolehellycomputecommonfactors
function parkinsoncolehellyfractionsubresolution
function parkinsoncolehellyprobabilitybound
subroutine parkinsoncolehellysubresolutionhypergeometrictabulate
subroutine parkinsoncolehellyupperboundhypergeometrictabulate
function conditionalmfbinweights2dintegrand
function particulatemassintegrand
subroutine particulateoperate
function luminosityintegrand
function multivariatenormalstochasticevaluate
function posteriorasprioreevaluate
function environmentaloverdensityconstructorinternal
function icmszextract
function integrandluminosityxray
function densitycontrastsroot
function lmnstystllrchrltfll2000extract
function velocitydispersionlambdarintegrand1
function velocitydispersionlambdarintegrand2
function velocitydispersionsolidangleincylinder
subroutine fftlog
module numerical_constants_physical
module numerical_constants_units
function node_component_disk_standard_density
function node_component_disk_standard_surface_density
function node_component_spheroid_standard_density

function node_component_spheroid_standard_rotation_curve_gradient
subroutine node_component_dynamics_statistics_bars_record
function node_component_hot_halo_cold_mode_rotation_curve_gradient_task
function intergalacticbackgroundinternalupdate
function jiang2014distributionvelocitytangential
function spincorrelatedorbit
function randomisotropicvelocity
function gnedin1999heatingrate
function creasey2012outflowrate
function kennicuttschmidtrate
function sohalofindererrorfractional
function cauchycumulative
function cauchyinverse
function voightdensity
function peakbackgrounddensity
function chabrier2001constructorinternal
function rootvariance
function filteredpowerrootvariancelogarithmicgradient
function variancepeakbackgroundsplitrootvariancelogarithmicgradient
function kitayamasuto1996gradienttime
function fixedconstructorparameters
function linearbarrierrate
function lognormalconstructorinternal
function normalconstructorinternal
function bhattacharya2011differential
function despali2015a
function despali2015p
function shethtormena
function shethtormennormalization
function errorconvolvedconvolution
function peacockdodds1996value
function lagrangianchan2017constructorinternal
function node_component_disk_very_simple_sfr
subroutine node_component_hot_halo_cold_mode_outflow_return
subroutine node_component_satellite_orbiting_rate_compute
function chandrasekhar1943acceleration
function isotropicorbit
function randomisotropicposition
function sphericalsymmetrytidaltensorradial
function zentner2005masslossrate
function blitz2006rate
function intgrtdsurfacedensitytimescale
function cauchyconstructorprobability
function cauchydensity
function voightcumulative
function normaldensity
subroutine statistics_points_power_spectrum
subroutine filteredpowerretabulate
subroutine filteredpowerrootvarianceandlogarithmicgradient
subroutine variancepeakbackgroundsplitrootvarianceandlogarithmicgradien
function variancepeakbackgroundsplitsigma8
function kitayamasuto1996value
function linearbarrierprobability
function convergencevarianceintegrand
function lognormalenvironmentmass
function normalenvironmentmass
function bhattacharya2011normalization
function despali2015normalization
function despali2015x
function shethtormendifferential
function shethtormenp
function fofbiasdifferential
function standardpowerdimensionless
function lagrangianchan2017value

function sharpkspaceconstructorinternal	function smoothkspaceconstructorinternal
function smoothkspacevalue	function tophatvalue
function tophatsharpkhybridconstructorinternal	subroutine tophatsharpkhybridradii
function bbkswdmhalfmodemass	function bode2001halfmodemass
function bryannorman1998densitycontrast	function bryannorman1998densitycontrastrateofchange
function kitayamasuto1996densitycontrast	function kitayamasuto1996densitycontrastrateofchange
function friendsoffriendsdensitycontrast	function friendsoffriendsdensitycontrastrateofchange
function haloradiusrootfunction	function virial_density_contrast_percolation_solver
function catalogprojectedcorrelationfunctionconstructorparameters	subroutine excursionsetsperform
function massfunctioncovarianceangularpowerradialtransf	subroutine massfunctioncovariancelssangularspectrum
subroutine massfunctioncovariancelsswindowfunction	subroutine massfunctioncovarianceperform
program tests_cosmic_age	program test_dark_matter_profiles_heated
program test_coordinate_systems	program test_integration
program test_mass_distributions	program tests_power_spectrum
program tests_sigma	program tests_spherical_collapse_dark_energy_eds
program tests_spherical_collapse_dark_energy_omega_half	program tests_spherical_collapse_dark_energy_omega_two_thirds
program tests_spherical_collapse_dark_energy_lambda	program tests_spherical_collapse_dark_energy_open
program tests_spherical_collapse_flat	program tests_spherical_collapse_open
function simpleratemassloss	function simpleratemassloss
function intergalacticmediumstateevolveconstructorinternal	function intergalacticmediumstateevolveodes
function intergalacticmediumstateevolveupdate	

file: numerical.constants.physical.F90

Description: Contains a module of useful physical constants.
Code lines: 69

module: numerical_constants_physical

Description: Contains various useful physical constants.
Code lines: 47
Contained by: file numerical.constants.physical.F90
Modules used: fgsl numerical_constants_math numerical_constants_prefixes
Used by: function bondi_hoyle_lyttleton_accretion_radius function bondi_hoyle_lyttleton_accretion_rate

function coldmodecoldmodefraction
function adafpowerjet
function shakurasunyaevpowerjet
function atomic_cross_section_compton
function sutherland1998total
function verner1996rate
function standardgrowthrate
function adafconstructorinternal
function eddingtonlimitedpowerjet
function filespectrum
function vernercrosssection
function vanhoof2014total
function volonteri2003separationinitial
function black_hole_eddington_accretion_rate
function black_hole_gravitational_radius
function black_hole_isco_specific_angular_momentum
function black_hole_isco_specific_energy_node
function hydrogennetworkcrosssection_h2_gamma_to_2h
function hydrogennetworkcrosssection_h2_gamma_to_h2plus_electron
function hydrogennetworkcrosssection_h2plus_gamma_to_2hplus_electron
function hydrogennetworkcrosssection_h2plus_gamma_to_h_hplus
function hydrogennetworkcrosssection_hminus_gamma_to_h_electron
function hydrogennetworkh_hminus_to_h2_electron_ratecoefficient
function hydrogennetworkhminus_electron_to_h_2electron_ratecoefficient
subroutine hydrogennetworkrateh2_gamma_to_2h
subroutine hydrogennetworkrateh2_gamma_to_h2plus_electron
subroutine hydrogennetworkrateh2_gamma_to_h2star_to_2h
subroutine hydrogennetworkrateh2_h_to_3h
subroutine hydrogennetworkrateh2_hplus_to_h2plus_h
subroutine hydrogennetworkrateh2plus_gamma_to_2hplus_electron
subroutine hydrogennetworkrateh2plus_gamma_to_h_hplus
subroutine hydrogennetworkrateh_gamma_to_hplus_electron
subroutine hydrogennetworkrateh_hplus_to_h2plus_photon
subroutine hydrogennetworkratehminus_gamma_to_h_electron
subroutine hydrogennetworkratehminus_h_to_2h_electron
subroutine hydrogennetworkratehminus_hplus_to_h2plus_electron
function cmbcomptoncoolingfunction
function simpletime
function constantrotationangularmomentumspecific
function matterlambdacomovingdistanceintegrand
function simpledensitycritical
function simpleomegaradiation
function virialdensitycontrastdefinitionvirialtemperature
function virialdensitycontrastdefinitionvirialvelocity
function dark_matter_halo_angular_momentum
function dark_matter_halo_spin
subroutine adiabaticgnedin2004computefactors
function genericcircularvelocitynumerical
function genericenergynumerical
function genericradialvelocitydispersionnumerical
function integrandpotential
function twobodyrelaxationspecificenergy

function twobodyrelaxationspecificenergygradient
function dark_matter_profile_mass_definition
function dark_matter_profile_rotation_curve_gradient_task
function dark_matter_profile_rotation_curve_task
function burkertcircularvelocity
function burkertcircularvelocitymaximum
function einastocircularvelocity
function einastocircularvelocitymaximum
function einastofreefallradius
function einastofreefallradiusincreaserate
function einastopotential
function einastoradiusfromspecificangularmomentum
function nfwcircularvelocity
function nfwcircularvelocitymaximum
function heatedcircularvelocity
function heateddensity
function heatedradiusenclosingmass
function heatedradiusinitialroot
subroutine efstathiou1982timescale
subroutine stabletimescale
subroutine radiussolve
function velocity_dispersion_integrand
function hothalomassdistributionrotationcurve
function hothalomassdistributionrotationcurvegradient
function font2008radiussolver
function rampressureaccelerationtimescale
function enzohydrostatictemperature
function gnedin2000odes
function intergalacticmediumstateelectronscatteringintegrand
function miyamotonagaipotential
function miyamotonagairotationcurve
function miyamotonagairotationcurvegradient
function exponentialdiskpotential
function exponentialdiskrotationcurve
function exponentialdiskrotationcurvegradient
function hernquistpotential
function sersicpotential
function betaprofilepotential
subroutine readassignspinparameters
function particulatepotentialintegrand
subroutine rotationcurveoperate
subroutine selfboundoperate
subroutine velocitydispersionoperate
function integrandcomptiony
function icmxrayluminosityextract
function lightconeextract
function lmnstyemssnlineextract
function velocitydispersionvelocitydensityintegrand
module numerical_constants_atomic
module chemical_structures
function kepler_orbits_energy
subroutine kepler_orbits_propagate
function kepler_orbits_velocity_scale
function node_component_disk_standard_potential
function node_component_disk_standard_rotation_curve
function node_component_disk_standard_rotation_curve_gradient
function node_component_spheroid_standard_potential
function node_component_spheroid_standard_rotation_curve
function node_component_spheroid_standard_rotation_curve_gradient
subroutine node_component_black_hole_noncentral_triple_interaction

subroutine node_component_black_hole_simple_rate_compute
function node_component_black_hole_simple_potential
function node_component_black_hole_simple_rotation_curve
function node_component_black_hole_simple_rotation_curve_gradient
subroutine node_component_black_hole_standard_rate_compute
function node_component_black_hole_standard_potential
function node_component_black_hole_standard_rotation_curve
function node_component_black_hole_standard_rotation_curve_gradient
subroutine node_component_disk_standard_radius_solver
function node_component_hot_halo_cold_mode_rotation_curve_gradient_task
function node_component_hot_halo_cold_mode_rotation_curve_task
subroutine node_component_satellite_orbiting_rate_compute
module node_component_spin_vitvitska
function radiationfieldintegrateovercrosssection_

function intergalacticbackgroundinternalflux
function intergalacticbackgroundinternalodes
function intergalacticbackgroundinternalupdate
function simpleratemassloss
function simpleratemassloss
function chandrasekhar1943acceleration
subroutine cole2000get
subroutine simpleget
subroutine standardget
subroutine cole2000get
subroutine covington2008get
function satellite_orbit_convert_to_current_potential
subroutine satellite_orbit_extremum_phase_space_coordinates
function sphericalsymmetrytidaltensorradial
function gnedin1999heatingrate
function zentner2005masslossrate
function zentner2005tidalradiussolver
function blitz2006constructorinternal
function blitz2006rate
function kennicuttschmidtrate
function convergencevarianceintegrand
function emptybeamconvergenceintegrand
program test_accretion_disks
program test_cooling_functions
program tests_kepler_orbits
function ideal_gas_jeans_length
function ideal_gas_sound_speed
function blackbody_emission
function simpleratemassloss
function simpleratemassloss
function intergalacticmediumstateevolveodes
function intergalacticmediumstateevolveupdate

file: numerical.constants.prefixes.F90
Description: Contains a module of useful numerical prefixes.
Code lines: 49

module: numerical_constants_prefixes
Description: Contains useful numerical prefixes.
Code lines: 27
Contained by: file numerical.constants.prefixes.F90
Used by: program xray_absorption_ism_wilms2000
function atomic_cross_section_compton
function coldmodecoldmodefraction
subroutine molecularhydrogengallipallacommonfactors

function molecularhydrogengallipallacoolingfunctiontemperaturelogslope
function twobodyrelaxationspecificenergy
function colordistributionsdssconstructorinternal
function rampressureaccelerationtimescale
subroutine sussingasciiload
subroutine sussinghdf5load
function gadgetbinaryimport
function integrandcomptiony
function integrandluminosityxray
function lightconeconstructorinternal
function lmnstyemssnlineextract
function rotationcurveunitsinsi
function velocitydispersionunitsinsi
function virialpropertiesunitsinsi
module numerical_constants_units
function node_component_disk_standard_-rotation_curve_gradient
subroutine node_component_hot_halo_-cold_mode_outflow_return
subroutine node_component_satellite_-orbiting_rate_compute
function intergalacticbackgroundinternalupdate
subroutine satellite_orbit_extremum_-phase_space_coordinates
function zentner2005masslossrate
function outflowrateintegrand
function kennicuttschmidtconstructorinternal
function extendedschmidtconstructorinternal
function convergencevarianceintegrand
subroutine halomodelgenerateperform
program test_accretion_disks
function intergalacticmediumstateevolveupdate
subroutine iratewritehalos
function simplehubbleconstant
subroutine efstathiou1982timescale
function galaxysizessdssconstructorinternal
function enzohydrostatictemperature
subroutine sussingtreeindicesread
subroutine galacticusimport
function gadgethdf5import
function icmxrayluminosityextract
function icmxrayluminosityunitsinsi
function lightconeextract
function lmnstystllrchrltfll2000extract
function satelliteorbitalextremaunitsinsi
function velocitymaximumunitsinsi
module numerical_constants_physical
subroutine store_unit_attributes_irate
function node_component_spheroid_-standard_rotation_curve_gradient
subroutine node_component_hot_halo_-standard_outflow_return
subroutine node_component_satellite_-orbiting_scale_set
function chandrasekhar1943acceleration
function gnedin1999heatingrate
function zentner2005tidalradiussolver
function blitz2006constructorinternal
subroutine krumholz2009computefactors
function standardconstructorparameters
function emptybeamconvergenceintegrand
function mergertreefilebuilderconstructorparameters
function intergalacticmediumstateevolveodes
subroutine iratereadhalos

file: numerical.constants.units.F90
Description: Contains a module of useful unit conversions.
Code lines: 54

module: numerical_constants_units
Description: Contains various useful unit conversions.
Code lines: 32
Contained by: file numerical.constants.units.F90
Modules used: fgsl numerical_constants_math numerical_constants_prefixes
Used by:

program xray_absorption_ism_wilms2000	subroutine hopkins2007buildfile
function atomic_cross_section_compton	function vernercrosssection
function sutherland1998total	function vanhoof2014total
function verner1996rate	function hydrogennetworkcrosssection_h2_gamma_to_2h
function hydrogennetworkcrosssection_h2_gamma_to_h2plus_electron	function hydrogennetworkcrosssection_h2plus_gamma_to_2hplus_electron
function hydrogennetworkcrosssection_h2plus_gamma_to_h_hplus	function hydrogennetworkcrosssection_hminus_gamma_to_h_electron
function hydrogennetworkh_hminus_to_h2_electron_ratecoefficient	function hydrogennetworkhminus_electron_to_h_2electron_ratecoefficient
subroutine hydrogennetworkrateh2_gamma_to_2h	subroutine hydrogennetworkrateh2_gamma_to_h2plus_electron
subroutine hydrogennetworkrateh2_gamma_to_h2star_to_2h	subroutine hydrogennetworkrateh2_h_to_3h
subroutine hydrogennetworkrateh2_hplus_to_h2plus_h	subroutine hydrogennetworkrateh2plus_gamma_to_2hplus_electron
subroutine hydrogennetworkrateh2plus_gamma_to_h_hplus	subroutine hydrogennetworkrateh_gamma_to_hplus_electron
subroutine hydrogennetworkrateh_hplus_to_h2plus_photon	subroutine hydrogennetworkratehminus_gamma_to_h_electron
subroutine hydrogennetworkratehminus_h_to_2h_electron	subroutine hydrogennetworkratehminus_hplus_to_h2plus_electron
function cmbcomptoncoolingfunction	function luminosityfunctionhalphaconstructorinternal
function bernardi2013sdsspointincluded	function gunawardhana2013sdssdistancemaximum
subroutine filter_response_load	function icmxrayluminosityextract
function icmxrayluminosityunitsinsi	function lmnstyemssnlineunitsinsi
module numerical_constants_astronomical	module numerical_constants_atomic
function radiationfieldintegrateovercrosssection_	function blackbodyflux

function intergalacticbackgroundinternalflux	function intergalacticbackgroundinternalodes
function intergalacticbackgroundinternalupdate	function standardconstructorparameters
function calzetti2000attenuation	function cardelli1989attenuation
function wittgordon2003constructorinternal	function tabulatedattenuation
program test_cooling_functions	function blackbody_emission
function intergalacticmediumstateevolveconstructorinternal	function intergalacticmediumstateevolveodes
function intergalacticmediumstateevolveupdate	

file: `numerical.differentiation.F90`
Description: Contains a module which performs numerical differentiation.
Code lines: 102

module: `numerical_differentiation`
Description: Implements numerical differentiation.
Code lines: 80
Contained by: file numerical.differentiation.F90
Modules used: interface_gsl iso_c_binding
Used by: function genericdensitylogslopenumerical; function genericenergygrowthratenumerical; function genericfreefallradiusincreaseratenumerical; program test_differentiation

interface: `differentiator`
Description: Constructors for the numerical derivatives class.
Code lines: 3
Contained by: module numerical_differentiation

function: `differentiatorconstructorinternal`
Description: Constructor for the numerical derivative class. Must be passed the function `f` for which derivatives will be computed.
Code lines: 9
Contained by: module numerical_differentiation

function: `differentiatorderivative`
Description: Compute a numerical derivative of the function at `x`. The initial stepsize is `h`. If present, the absolute error estimate is returned in `errorAbsolute`.
Code lines: 15
Contained by: module numerical_differentiation
Modules used: galacticus_error

subroutine: `differentiatordestructor`
Description: Destructor for the numerical derivative class.
Code lines: 7
Contained by: module numerical_differentiation

file: `numerical.fftw3.F90`
Description: Contains a module which imports the FFTW3 library Fortran interface.
Code lines: 48

module: `fftw3`
Description: Imports the FFTW3 library Fortran interface.
Code lines: 26
Contained by: file `numerical.fftw3.F90`
Modules used: `iso_c_binding`
Used by: subroutine `fullskywindowfunctions`; subroutine `randompointswindowfunctions`; subroutine `statistics_points_power_spectrum`; subroutine `massfunctioncovariancelsswindowfunction`

function: `fftw_wavenumber`
Description: Return the wavenumber (in units of $1/L$ where L is the box length) corresponding to element `k` out of `n` of a 1-D FFT using the FFTW convention.
Code lines: 14
Contained by: module `fftw3`

file: `numerical.integration.F90`
Description: Contains a module which performs numerical integration.
Code lines: 172

module: `numerical_integration`
Description: Implements numerical integration.
Code lines: 150
Contained by: file `numerical.integration.F90`
Modules used: `fgsl`
Used by: subroutine `matterlambdamakedistancetable`; subroutine `nbodyerrorstabulate`; function `genericenclosedmassdifferencenumerical`; function `genericenergynumerical`; function `genericrickspacenumerical`; function `genericpotentialdifferencenumerical`; function `genericpotentialnumerical`; function `genericradialmomentnumerical`; function `genericradialvelocitydispersionnumerical`; function `rootradiusfreefall`; function `burkertfreefalltimescalefree`; function `burkertprofileenergy`; function `burkertradialvelocitydispersionscalefree`; subroutine `einastoenergytablemake`; subroutine `einastofourierprofiletablemake`; function `einastofreefalltimescalefree`; function `einastoradialvelocitydispersionscalefree`; function `nfwfreefalltimescalefree`; function `nfwprofileenergy`; function `galactic_structure_velocity_dispersion`

function disksizeinclntnintegrandx
function grvtnllnsngoperatedistribution
subroutine halo_model_projected_correlation
function enzohydrostaticenclosedmass
subroutine intergalacticmediumstateelectronscattering
subroutine miyamotonagaisurfacedensitytabulate
subroutine sersictabulate
function parkinsoncolehellymassbranchrootderivative
function generalizedpressschechterfractionsubresolution
function generalizedpressschechterprobability
subroutine sampleddistributionconstruct
function particulatesmoothingintegrandz
function sedfitevaluate
function spindistributionevaluate
function icmxrayluminosityextract
function velocitydispersionextract
function velocitydispersionlineofsightvelocitydispersionintegrand
function radiationfieldintegrateovercrosssection_integrand
function jiang2014constructorinternal
function intgrtdsurfacedensitytimescale
subroutine stellar_population_luminosity_tabulate
function zhanghuig2integrated
function environmentaveragedifferential
function radius_root
function tcollapse
subroutine halomassfunctionperform

function disksizeinclntnroot
function spinnbodyerrorsoperatescalar
function enzohydrostaticdensitynormalization
function enzohydrostaticradialmoment
subroutine miyamotonagaimassenclosedtabulate
function sphericalmassenclosedbysphere
function parkinsoncolehellymassbranchroot
function parkinsoncolehellyprobability
function generalizedpressschechtermassbranchroot
function stellarmassfunctionsample
function conditionalmfbinweights2d
subroutine particulatetabulateenergydistribution
function massfunctionevaluate
function icmszextract
function projecteddensityextract
function velocitydispersionlambdarintegrand1
function table_logarithmic_integration_weights
function intergalacticbackgroundinternalupdate
function creasey2012outflowrate
function standardenergyinputcumulative
function rootvariance
subroutine takahashi2011lensingdistributionconstruct
function errorconvolveddifferential
subroutine spherical_collapse_matter_lambda_nonlinear_mapping
subroutine conditionalmassfunctionperform
subroutine massfunctioncovariancecomputevolumenormalizations

function massfunctioncovariancegalaxyrootpowerspectrum
function massfunctioncovariancehalooccupancytimeinterand
subroutine massfunctioncovariancelssangularspectrum
function massfunctioncovariancemassfunctiontimeintegrandi
subroutine massfunctioncovarianceperform
subroutine powerspectraperform
program test_integration
function integrand4
program test_integration2
function intergalacticmediumstateevolveodes
function intergalacticmediumstateevolveupdate

function: `integrandwrapper`

Description: Wrapper function used for GSL integration functions.
Code lines: 11
Contained by: module numerical_integration
Modules used: `iso_c_binding`

function: `integrate`

Description: Integrates the supplied `integrand` function.
Code lines: 90
Contained by: module numerical_integration
Modules used: galacticus_error `iso_c_binding`

subroutine: `integrate_done`

Description: Frees up integration objects that are no longer required.
Code lines: 9
Contained by: module numerical_integration

subroutine: `integration_gsl_error_handler`

Description: Handle errors from the GSL library during integration.
Code lines: 9
Contained by: module numerical_integration
Modules used: `iso_c_binding`

file: `numerical.integration2.F90`

Description: Contains a module which implements a variety of numerical integrators.
Code lines: 1829

module: `numerical_integration2`

Description: Implements a variety of numerical integrators.
Code lines: 1807
Contained by: file numerical.integration2.F90
Used by: subroutine standardevolve
subroutine odeiv2_solve
function standardinterpolate
program test_ode_solver
program test_initial_mass_functions
program test_integration2
module test_integration2_functions

function: `adaptivecompositetrapezoidalevaluate1d`
Description: Evaluate a one-dimension integral using a numerical composite trapezoidal rule.
Code lines: 104
Contained by: module `numerical_integration2`
Modules used: `galacticus_error` `numerical_comparison`

function: `compositegausskronrod1devaluate`
Description: Evaluate a one-dimension integral using a numerical composite Gauss-Kronrod rule.
Code lines: 107
Contained by: module `numerical_integration2`
Modules used: `galacticus_error` `numerical_comparison`

subroutine: `compositegausskronrod1devaluateinterval`
Description: Evaluate the integral over an interval using the Gauss-Kronrod method (also estimates the error). Specific implementation is based on that in the GSL.
Code lines: 64
Contained by: module `numerical_integration2`

subroutine: `compositegausskronrod1dinitialize`
Description: Initialize a one-dimensional, composite Gauss-Kronrod numerical integrator. Evaluation points and weights are taken from those used in the GSL.
Code lines: 31
Contained by: module `numerical_integration2`
Modules used: `galacticus_error` `memory_management`

function: `compositetrapezoidalevaluate1d`
Description: Evaluate a one-dimension integral using a numerical composite trapezoidal rule.
Code lines: 30
Contained by: module `numerical_integration2`
Modules used: `galacticus_error` `numerical_comparison`

subroutine: `compositetrapezoidalinitialize1d`
Description: Initialize a one-dimensional, composite trapezoidal numerical integrator.
Code lines: 10
Contained by: module `numerical_integration2`
Modules used: `galacticus_error`

subroutine: `integrandmulti1dset`
Description: Initialize the integrand for one-dimensional numerical integrators.
Code lines: 10
Contained by: module `numerical_integration2`

subroutine: `integrandmultivectorizedset1d`
Description: Initialize the integrand for one-dimensional numerical integrators.
Code lines: 10
Contained by: module `numerical_integration2`

subroutine: `integrandset1d`
Description: Initialize the integrand for one-dimensional numerical integrators.
Code lines: 8
Contained by: module `numerical_integration2`

subroutine: `integrandvectorizedset1d`
Description: Initialize the integrand for one-dimensional numerical integrators.
Code lines: 8
Contained by: module `numerical_integration2`

type: `integrator`
Description: Generic numerical integrator class.
Code lines: 14
Contained by: module `numerical_integration2`

type: `integrator1d`
Description: Generic one-dimensional numerical integrator class.
Code lines: 22
Contained by: module `numerical_integration2`

type: `integratoradaptivecompositetrapezoidal1d`
Description: One-dimensional numerical integrator class using an adaptive composite trapezoidal rule.
Code lines: 5
Contained by: module `numerical_integration2`

type: `integratorcompositegausskronrod1d`
Description: One-dimensional numerical integrator class using a composite Gauss-Kronrod rule.
Code lines: 24
Contained by: module `numerical_integration2`

type: `integratorcompositetrapezoidal1d`
Description: One-dimensional numerical integrator class using a composite trapezoidal rule.
Code lines: 16
Contained by: module `numerical_integration2`

type: `integratormulti`
Description: Generic numerical integrator class.
Code lines: 15
Contained by: module `numerical_integration2`

type: `integratormulti1d`
Description: Generic one-dimensional multi-integrand numerical integrator class.
Code lines: 23
Contained by: module `numerical_integration2`

subroutine: `integratormultidestructor`
Description: Destructor for the `integratorMulti` class.
Code lines: 9

Contained by: module numerical_integration2
Modules used: memory_management

type: integratormultivectorized1d
Description: Generic one-dimensional multi-integrand vectorized numerical integrator class.
Code lines: 17
Contained by: module numerical_integration2

type: integratormultivectorizedcompositegausskronrod1d
Description: One-dimensional multi-integrand numerical integrator class using a vectorized composite Gauss-Kronrod rule.
Code lines: 30
Contained by: module numerical_integration2

type: integratormultivectorizedcompositetrapezoidal1d
Description: One-dimensional multi-integrand numerical integrator class using a vectorized composite trapezoidal rule.
Code lines: 22
Contained by: module numerical_integration2

type: integratorvectorized1d
Description: Generic one-dimensional vectorized numerical integrator class.
Code lines: 16
Contained by: module numerical_integration2

type: integratorvectorizedcompositegausskronrod1d
Description: One-dimensional numerical integrator class using a vectorized composite Gauss-Kronrod rule.
Code lines: 30
Contained by: module numerical_integration2

type: integratorvectorizedcompositetrapezoidal1d
Description: One-dimensional numerical integrator class using a vectorized composite trapezoidal rule.
Code lines: 23
Contained by: module numerical_integration2

type: interval
Code lines: 5
Contained by: module numerical_integration2

type: intervalmulti
Code lines: 6
Contained by: module numerical_integration2

type: intervalmultilist
Code lines: 2
Contained by: module numerical_integration2

function: multivectorizedcompositegausskronrod1devaluate
Description: Evaluate a one-dimension integral using a numerical composite Gauss-Kronrod rule.

Code lines: 182
Contained by: module numerical_integration2
Modules used: galacticus_error iso_c_binding numerical_comparison sort

subroutine: multivectorizedcompositegausskronrod1devaluateinterval
Description: Evaluate the integral over an interval using the Gauss-Kronrod method (also estimates the error). Specific implementation is based on that in the GSL.
Code lines: 61
Contained by: module numerical_integration2

subroutine: multivectorizedcompositegausskronrod1dinitialize
Description: Initialize a one-dimensional, multi-integrand, vectorized composite Gauss-Kronrod numerical integrator. Evaluation points and weights are taken from those used in the GSL.
Code lines: 31
Contained by: module numerical_integration2
Modules used: galacticus_error memory_management

function: multivectorizedcompositetrapezoidal1devaluate
Description: Evaluate a one-dimension integral using a numerical composite trapezoidal rule.
Code lines: 365
Contained by: module numerical_integration2
Modules used: galacticus_display galacticus_error iso_c_binding iso_varying_string numerical_comparison sort

subroutine: multivectorizedcompositetrapezoidal1dinitialize
Description: Initialize a one-dimensional, multi-integrand, vectorized composite trapezoidal numerical integrator.
Code lines: 11
Contained by: module numerical_integration2
Modules used: galacticus_error memory_management

subroutine: tolerancesetgeneric
Description: Initialize the tolerances for numerical integrators.
Code lines: 19
Contained by: module numerical_integration2
Modules used: galacticus_error

subroutine: tolerancessetgeneric
Description: Initialize the tolerances for multi-integrand numerical integrators.
Code lines: 32
Contained by: module numerical_integration2
Modules used: galacticus_error memory_management

function: vectorizedcompositegausskronrod1devaluate
Description: Evaluate a one-dimension integral using a numerical composite Gauss-Kronrod rule.
Code lines: 107
Contained by: module numerical_integration2

Modules used: galacticus_error numerical_comparison

subroutine: vectorizedcompositegausskronrod1devaluateinterval
Description: Evaluate the integral over an interval using the Gauss-Kronrod method (also estimates the error). Specific implementation is based on that in the GSL.
Code lines: 55
Contained by: module numerical_integration2

subroutine: vectorizedcompositegausskronrod1dinitialize
Description: Initialize a one-dimensional, vectorized composite Gauss-Kronrod numerical integrator. Evaluation points and weights are taken from those used in the GSL.
Code lines: 31
Contained by: module numerical_integration2
Modules used: galacticus_error memory_management

function: vectorizedcompositetrapezoidalevaluate1d
Description: Evaluate a one-dimension integral using a numerical vectorized composite trapezoidal rule.
Code lines: 43
Contained by: module numerical_integration2
Modules used: galacticus_error iso_c_binding
numerical_comparison yepcore
yepmath

subroutine: vectorizedcompositetrapezoidalinitialize1d
Description: Initialize a one-dimensional, vectorized composite trapezoidal numerical integrator.
Code lines: 23
Contained by: module numerical_integration2
Modules used: galacticus_error memory_management

file: numerical.interpolation.2D.irregular.F90
Description: Contains a module which implements a convenient interface to the BIVAR 2D interpolation on irregularly spaced points package.
Code lines: 135

module: numerical_interpolation_2d_irregular
Description: Implements a convenient interface to the BIVAR 2D interpolation on irregularly spaced points package.
Code lines: 112
Contained by: file numerical.interpolation.2D.irregular.F90
Used by: file stellar_astrophysics.file.F90 program test_interpolation_2d

type: interp2dirregularobject
Code lines: 3
Contained by: module numerical_interpolation_2d_irregular

interface: interpolate_2d_irregular
Code lines: 3
Contained by: module numerical_interpolation_2d_irregular

function: `interpolate_2d_irregular_array`

Description: Perform interpolation on a set of points irregularly spaced on a 2D surface.
Code lines: 73
Contained by: module `numerical_interpolation_2d_irregular`
Modules used: `bivar` `memory_management`

function: `interpolate_2d_irregular_scalar`

Description: Perform interpolation on a set of points irregularly spaced on a 2D surface. This version is simply a wrapper that does look up for a scalar point by calling the array-based version.
Code lines: 16
Contained by: module `numerical_interpolation_2d_irregular`

file: `numerical.interpolation.F90`

Description: Contains a module which acts as a simple interface to the GNU Scientific Library interpolation routines.
Code lines: 313

module: `numerical_interpolation`

Description: A simple interface to the GNU Scientific Library interpolation routines.
Code lines: 290
Contained by: file `numerical.interpolation.F90`
Modules used: `fgsl` `iso_c_binding`
Used by:

- subroutine `filedestructor`
- subroutine `ciefiledestructor`
- subroutine `atomiccieclouddestructor`
- subroutine `ciefileinterpolatingfactors`
- function `matterdarkenergycosmictime`
- function `matterlambdacosmictime`
- function `matterlambdadistanceangular`
- function `matterlambdadistancecomovingconvert`
- function `matterlambdaexpansionfactor`
- subroutine `matterlambdamakeexpansionfactortable`
- function `staticuniversedistanceangular`
- subroutine `einastodestructor`
- function `einastoenergygrowthrate`
- subroutine `einastofourierprofiletablemake`
- function `einastofreefallradiusincreaserate`
- function `einastokspace`
- function `filespectrum`
- subroutine `ciefileinterpolatingfactors`
- subroutine `ciefiledestructor`
- subroutine `atomiccieclouddestructor`
- subroutine `matterdarkenergymakeexpansionfactortable`
- subroutine `matterlambdadestructor`
- function `matterlambdadistancecomoving`
- function `matterlambdadistanceluminosity`
- subroutine `matterlambdamakedistancetable`
- function `matterlambdatimeatdistancecomoving`
- function `staticuniversedistanceluminosity`
- function `einastoenergy`
- subroutine `einastoenergytablemake`
- function `einastofreefallradius`
- subroutine `einastofreefalltabulate`
- function `einastoradialvelocitydispersion`

subroutine einastoradialvelocitydispersiontabulate
function einastoradiusfromspecificangularmomentumscalefree
subroutine einastoradiusfromspecificangularmomentumtablemake
function filter_response
function fileelectronfraction
function fileneutralheliumfraction
function fileneutralhydrogenfraction
function filesinglyionizedheliumfraction
function filetemperature
function internalelectronfraction
function internalneutralheliumfraction
function internalneutralhydrogenfraction
function internalsinglyionizedheliumfraction
function internaltemperature
function sersicdensity
function sersicmassenclosedbysphere
function sersicpotential
subroutine sersictabulate
subroutine sampleddistributionconstruct
subroutine historystore
function historytimeevolveto
subroutine recordevolutionstore
function recordevolutiontimeevolveto
subroutine exportoperate
subroutine regridtimesoperate
subroutine lmnstyemssnlinedestructor
function lmnstyemssnlineextract
subroutine history_interpolated_increment
function positionpresetposition
function positionpresetvelocity
function satellitepresetmergeboundmass
function satellitepresetnodeindex
subroutine node_component_dynamics_statistics_bars_record
subroutine table_2d_linlinlin_destroy
function table_2d_linlinlin_interpolate
subroutine table_generic_1d_destroy
function table_generic_1d_interpolate
function table_generic_1d_interpolate_gradient
function intergalacticbackgroundfileflux
subroutine intergalacticbackgroundfiletimeset
function intergalacticbackgroundinternalflux
function hegerwoosley2002energycumulative
subroutine fileinterpolationcompute
function stellar_population_luminosity
subroutine stellar_population_luminosity_track
subroutine noninstantaneousrates
function fileluminosity
subroutine spectraltabledestructor1d
subroutine spectraltabledestructorscalar
function standardinterpolate
function barkana2001wdmgradientmass
function barkana2001wdmvalue
subroutine farahidestructor
subroutine farahifileread
function farahiprobability
function farahirate
function farahiratenoncrossing
subroutine farahiratetabulate
function farahimidpointprobability
subroutine farahimidpointratetabulate
function zhanghuiprobability
function zhanghuihighorderprobability
function tinker2008formdifferential
function cosmicemuvalue
function massfunctioncovariancelargescalestructureintegrand
program test_interpolation

function: `interpolate`
Description: Perform an interpolation of `x` into `xArray()` and return the corresponding value in `yArray()`.
Code lines: 100
Contained by: module numerical_interpolation
Modules used: galacticus_error iso_varying_string table_labels

function: `interpolate_derivative`
Description: Perform an interpolation of `x` into `xArray()` and return the corresponding first derivative of `yArray()`.
Code lines: 70
Contained by: module numerical_interpolation
Modules used: galacticus_error iso_varying_string table_labels

subroutine: `interpolate_done`
Description: Free interpolation objects when they are no longer required.
Code lines: 20
Contained by: module numerical_interpolation

function: `interpolate_linear_do`
Description: Given an array index `iInterpolate` and interpolating factors `interpolationFactors` for array `yArray`, return a linearly interpolated value.
Code lines: 11
Contained by: module numerical_interpolation
Modules used: kind_numbers

function: `interpolate_linear_generate_factors`
Description: Return interpolating factors for linear interpolation in the array `xArray()` given the index in the array which brackets value `x`.
Code lines: 13
Contained by: module numerical_interpolation
Modules used: kind_numbers

function: `interpolate_locate`
Description: Perform an interpolation of `x` into `xArray()` and return the corresponding value in `yArray()`.
Code lines: 53
Contained by: module numerical_interpolation
Modules used: galacticus_error kind_numbers

file: `numerical.meshes.F90`
Description: Contains a module which provides tools for working with grids.
Code lines: 109

module: `meshes`
Description: Provide tools for working with grids.
Code lines: 87
Contained by: file numerical.meshes.F90

Used by: subroutine martin2010alfalfarandomsinitialize; subroutine fullskywindowfunctions; subroutine randompointswindowfunctions; subroutine statistics_points_power_spectrum; program test_meshes

subroutine: meshes_apply_point
Description: Apply a point to a mesh.
Code lines: 60
Contained by: module meshes
Modules used: galacticus_error; iso_c_binding

function: triangular_shaped_cloud_integral
Description: Return the integral over a triangular shaped cloud given the fraction of the cloud length in a cell.
Code lines: 11
Contained by: module meshes

file: numerical.nearest_neighbors.F90
Description: Contains a module which wraps the ANN (Approximate Nearest Neighbor) library.
Code lines: 241

module: nearest_neighbors
Description: Wraps the ANN (Approximate Nearest Neighbor) library.
Code lines: 216
Contained by: file numerical.nearest_neighbors.F90
Modules used: iso_c_binding
Used by: file models.likelihoods.posterior_as_prior.F90; subroutine environmentaloverdensityoperate; subroutine paircountsoperate; subroutine statistics_points_correlation

interface: nearestneighbors
Code lines: 2
Contained by: module nearest_neighbors

subroutine: nearestneighborsclose
Description: Closes the ANN (Approximate Nearest Neighbor) library.
Code lines: 13
Contained by: module nearest_neighbors
Modules used: galacticus_error

function: nearestneighborsconstructor
Description: Constructs a nearest neighor search object.
Code lines: 18
Contained by: module nearest_neighbors
Modules used: galacticus_error; iso_c_binding

subroutine: nearestneighborsdestructor

Description: Destroys a nearest neighbor search object.
Code lines: 15
Contained by: module nearest_neighbors
Modules used: galacticus_error

subroutine: nearestneighborssearch
Description: Return indices and distances to the (approximate) nearest neighbors.
Code lines: 25
Contained by: module nearest_neighbors
Modules used: galacticus_error

subroutine: nearestneighborssearchfixedradius
Description: Return indices and distances to all neighbors within a given radius.
Code lines: 44
Contained by: module nearest_neighbors
Modules used: galacticus_error memory_management

file: numerical.points.F90
Description: Contains a module which provides tools for working with sets of points.
Code lines: 172

module: points
Description: Provide tools for working with sets of points.
Code lines: 150
Contained by: file numerical.points.F90
Used by: subroutine catalogprojectedcorrelationfunctionperform

subroutine: points_prune
Description: Prune a set of points.
Code lines: 21
Contained by: module points
Modules used: memory_management

subroutine: points_replicate
Description: Apply a simple translation to a set of points.
Code lines: 27
Contained by: module points
Modules used: memory_management

subroutine: points_rotate
Description: Apply a rotation to a set of points.
Code lines: 23
Contained by: module points
Modules used: vectors

subroutine: points_survey_geometry
Description: Select a set of points that lie within a given survey geometry
Code lines: 38

Contained by: module points
Modules used: galacticus_display geometry_surveys memory_management

subroutine: points_translate
Description: Apply a simple translation to a set of points.
Code lines: 24
Contained by: module points

file: numerical.random.F90
Description: Contains a module which implements pseudo-random numbers.
Code lines: 269

module: pseudo_random
Description: Implements pseudo-random numbers.
Code lines: 247
Contained by: file numerical.random.F90
Modules used: fgsl
Used by:

function bett2007sample	subroutine martin2010alfalfarandomsinitialize
subroutine fullskywindowfunctions	subroutine randompointswindowfunctions
module merger_tree_branching	function massbranchcdmassumptions
function massbranchgeneric	function generalizedpressschechtermassbranch
function buildconstruct	subroutine sampleddistributionpseudorandomsamplecmf
subroutine cole2000build	function fullyspecifiedconstruct
subroutine readbuildisolatedparentpointers	function readconstruct
module merger_tree_read_importers	file merger_trees.construct.read.importer.SussingMergerTrees
function smoothaccretionconstruct	subroutine particulateoperate
module merger_tree_state_store	file models.likelihoods.multivariate_normal.stochastic.F90
subroutine meanpositionoperate	subroutine paircountsoperate
subroutine rotationcurveoperate	subroutine selfboundoperate
subroutine velocitydispersionoperate	module galacticus_nodes
function annealeddifferentialevolutionacceptproposal	function differentialevolutionacceptproposal
function differentialevolutionchainselect	subroutine differentialevolutionsimulate
subroutine particleswarmsimulate	function stochasticdifferentialevolutionacceptproposal
function tempereddifferentialevolutionacceptproposal	subroutine latinhypercubeinitialize

function randomtimeuntilmerging
function randomisotropicvelocity
module statistics_distributions_-discrete
subroutine catalogprojectedcorrelationfunctionperform
program test_math_distributions
function randomisotropicposition
module statistics_distributions
function catalogprojectedcorrelationfunctionconstructorparameters
subroutine halomodelgenerateperform
program test_random

interface: pseudorandom
Description: Constructors for the pseudoRandom class.
Code lines: 3
Contained by: module pseudo_random

function: pseudorandomclone
Description: Clone a pseudo-random sequence object.
Code lines: 12
Contained by: module pseudo_random

function: pseudorandomconstructor
Description: Construct a pseudo-random sequence object.
Code lines: 9
Contained by: module pseudo_random
Modules used: iso_c_binding

subroutine: pseudorandomdestructor
Description: Destroy the pseudo-sequence wrapper classs.
Code lines: 7
Contained by: module pseudo_random

subroutine: pseudorandomfree
Description: Frees a pseudo-random sequence object.
Code lines: 7
Contained by: module pseudo_random

subroutine: pseudorandominitialize
Description: Initialize pseudo-random number sequence generators.
Code lines: 22
Contained by: module pseudo_random
Modules used: input_parameters

function: pseudorandomnormalsample
Description: Sample from a standard normal distribution.
Code lines: 11
Contained by: module pseudo_random

function: pseudorandompoissonsample
Description: Sample from a Poisson distribution with the given mean.
Code lines: 12
Contained by: module pseudo_random

subroutine: `pseudorandomreset`
Code lines: 36
Contained by: module `pseudo_random`
Modules used: `mpi`

subroutine: `pseudorandomrestore`
Description: Store a pseudo-random sequence object state to file.
Code lines: 14
Contained by: module `pseudo_random`

subroutine: `pseudorandomstore`
Description: Store a pseudo-random sequence object state to file.
Code lines: 11
Contained by: module `pseudo_random`

function: `pseudorandomuniformsample`
Description: Sample from a uniform distribution on the interval 0 to 1.
Code lines: 12
Contained by: module `pseudo_random`

file: `numerical.random.quasi.F90`
Description: Contains a module which implements quasi-random sequences.
Code lines: 107

module: `quasi_random`
Description: Implements quasi-random sequences.
Code lines: 85
Contained by: file `numerical.random.quasi.F90`
Modules used: `fgsl`
Used by: subroutine `sampleddistributionquasirandomsamplecmf`

subroutine: `quasi_random_free`
Description: Frees a quasi-random sequence object.
Code lines: 7
Contained by: module `quasi_random`

interface: `quasi_random_get`
Code lines: 3
Contained by: module `quasi_random`

function: `quasi_random_get_array`
Description: Returns an array giving a quasi-random points in a `quasiSequenceDimension`-dimensional space.
Code lines: 29
Contained by: module `quasi_random`

function: `quasi_random_get_scalar`

Description: Returns a scalar giving a quasi-random point in a 1-dimensional space.
Code lines: 29
Contained by: module quasi_random

file: numerical.ranges.F90
Description: Contains a module which implements construction of numerical ranges.
Code lines: 88

module: numerical_ranges
Description: Implements construction of numerical ranges.
Code lines: 66
Contained by: file numerical.ranges.F90
Used by:

program xray_absorption_ism_wilms2000	subroutine hopkins2007buildfile
subroutine matterdarkenergymakeexpansionfactortable	subroutine matterlambdamakedistancetable
subroutine matterlambdamakeexpansionfactortable	subroutine einastoenergytablemake
subroutine einastofourierprofiletablemake	subroutine einastofreefalltabulate
subroutine einastoradialvelocitydispersiontabulate	subroutine einastoradiusfromspecificangularmomentumtablemak
subroutine galacticus_meta_evolver_-profile	function hivshalomassrelationpadmanabhan2017constructorin
function localgroupmassfunctionconstructorinterna	function blackholebulgerelationconstructorinternal
function correlationfunctionconstructorinternal	function stellarvshalomassrelationleauthaud2012constructo
subroutine galacticus_output_halo_-model_initialize	subroutine halo_model_projected_-correlation
subroutine interface_cloudy_cie_-tabulate	subroutine sersictabulate
subroutine sampleddistributionconstruct	subroutine sampleddistributionuniformsamplecmf
function historyconstructorinternal	function recordevolutionconstructorinternal
function conditionalmfconstructorinternal	function profilerconstructorinternal
function regridtimesconstructorinternal	subroutine paircountsoperate
subroutine history_create	subroutine history_extend
subroutine history_increment	subroutine history_long_integer_create
subroutine history_timesteps	subroutine table_2dlogloglin_create
subroutine table_linear_1d_create	subroutine table_linear_cspline_1d_-create
subroutine table_linear_monotone_-cspline_1d_create	function intergalacticbackgroundinternalconstructorintern

subroutine insitumake	function metallicitysplitconstructorinternal
subroutine metallicitysplitmake	subroutine statistics_points_correlation
subroutine statistics_points_power_spectrum	subroutine noninstantaneoushistorycreate
function standardinterpolate	function farahiprobability
subroutine farahiratetabulate	function farahimidpointprobability
subroutine farahimidpointratetabulate	function zhanghuiprobability
function zhanghuihighorderprobability	subroutine takahashi2011lensingdistributionconstruct
subroutine spherical_collapse_matter_lambda_nonlinear_mapping	function conditionalmassfunctionconstructorinternal
subroutine excursionsetsperform	subroutine halomassfunctionperform
function halomodelprojectedcorrelationfunctionconstructorinternal	subroutine massfunctionintervariianceperform
subroutine powerspectraperform	program test_make_ranges
program tests_sigma	function intergalacticmediumstateevolveconstructorinternal

function: `make_range`

Description: Builds a numerical range between `rangeMinimum` and `rangeMaximum` using `rangeNumber` points and spacing as specified by `rangeType` (defaulting to linear spacing if no `rangeType` is given).
Code lines: 53
Contained by: module numerical_ranges
Modules used: galacticus_error

file: `numerical.root_finder.F90`

Description: Contains a module which does root finding.
Code lines: 610

module: `root_finder`

Description: Implements root finding.
Code lines: 589
Contained by: file numerical.root_finder.F90
Modules used: `fgsl` `iso_c_binding`
Used by:

function tidalradiusseparationinitial	function simpleradius
function matterdarkenergydominationepochmatter	function matterdarkenergyequalityepochmatterdarkenergy
function correa2015time	function nbodyerrorsnoncentralchisquaremode
function adiabaticgnedin2004radiusinitial	function adiabaticgnedin2004radiusinitialderivative
function genericcircularvelocitymaximumnumerical	function genericfreefallradiusnumerical
function genericradiusenclosingdensitynumerical	function genericradiusenclosingmassnumerical

function genericradiusfromspecificangularmomentumnumerical
function nfw1996concentration
subroutine ludlow2014formationtimerootfunctionset
subroutine ludlow2016formationtimerootfunctionset
function einastocircularvelocitymaximum
function heatedradiusinitial
function galactic_structure_radius_enclosing_mass
function font2008radiusstripped
function sphericalradiushalfmass
function massbranchgeneric
subroutine fixedmassconstruct
function densitycontrastsextract
function jiang2014orbit
function satellite_orbit_equivalent_circular_orbit_radius
function zentner2005masslossrate
function betainverse
subroutine baryonicmodifierrenormalize
subroutine make_table
function perturbation_maximum_radius
function virial_density_contrast_percolation_solver
program test_root_finding
function correa2015concentration
file dark_matter_profiles.structure.concentration.Schneider2015.F90
file dark_matter_profiles.structure.scale.Ludlow2016.F90
function concentrationradius
function einastoradiusenclosingdensity
function galactic_structure_radius_enclosing_density
function disksizeinclinationconstructorinternal
function instantreionizationigmconstructorinternal
subroutine sersictabulate
function generalizedpressschechtermassbranch
subroutine readassignscaleradii
subroutine cole2000get
file satellites.merging.virial_orbits.Wetzel2010.F90
subroutine satellite_orbit_extremum_phase_space_coordinates
function krumholz2009intervals
subroutine takahashi2011lensingdistributionconstruct
function environmentaverageddifferential
subroutine make_table
subroutine spherical_collapse_matter_lambda_nonlinear_mapping
subroutine halomodelgenerateperform

subroutine: `root_finder_derivative_type`
Description: Sets the type to use in a `rootFinder` object.
Code lines: 10
Contained by: module root_finder

subroutine: `root_finder_destroy`
Description: Destroy a root finder object.
Code lines: 16
Contained by: module root_finder

subroutine: `root_finder_finalize`
Description: Finalize a root finder object.

Code lines: 7
Contained by: module root_finder

function: root_finder_find
Description: Finds the root of the supplied root function.
Code lines: 231
Contained by: module root_finder
Modules used: galacticus_display galacticus_error iso_varying_string

subroutine: root_finder_gsl_error_handler
Description: Handle errors from the GSL library during root finding.
Code lines: 9
Contained by: function root_finder_find
Modules used: iso_c_binding

function: root_finder_is_initialized
Description: Return whether a rootFinder object is initalized.
Code lines: 7
Contained by: module root_finder

subroutine: root_finder_range_expand
Description: Sets the rules for range expansion to use in a rootFinder object.
Code lines: 48
Contained by: module root_finder
Modules used: galacticus_error

subroutine: root_finder_root_function
Description: Sets the function to use in a rootFinder object.
Code lines: 12
Contained by: module root_finder

subroutine: root_finder_root_function_derivative
Description: Sets the function to use in a rootFinder object.
Code lines: 16
Contained by: module root_finder

subroutine: root_finder_tolerance
Description: Sets the tolerances to use in a rootFinder object.
Code lines: 9
Contained by: module root_finder

subroutine: root_finder_type
Description: Sets the type to use in a rootFinder object.
Code lines: 10
Contained by: module root_finder

function: root_finder_wrapper_function
Description: Wrapper function callable by FGSL used in root finding.

Code lines: 20
Contained by: module root_finder

subroutine: root_finder_wrapper_function_both
Description: Wrapper function callable by FGSL used in root finding.
Code lines: 10
Contained by: module root_finder

function: root_finder_wrapper_function_derivative
Description: Wrapper function callable by FGSL used in root finding.
Code lines: 10
Contained by: module root_finder

type: rootfinder
Description: Type containing all objects required when calling the FGSL root solver function.
Code lines: 95
Contained by: module root_finder

type: rootfinderlist
Description: Type used to maintain a list of root finder objects when root finding is performed recursively.
Code lines: 5
Contained by: module root_finder

file: numerical.search.F90
Description: Contains a module which implements searching of ordered arrays.
Code lines: 243

module: arrays_search
Description: Implements searching of ordered arrays.
Code lines: 220
Contained by: file numerical.search.F90
Modules used: iso_c_binding kind_numbers
Used by:

function sutherland1998total	function vanhoof2014total
subroutine galacticus_meta_evolver_-profile	function identityoperatescalar
function squareisinlightcone	function squareposition
function squarereplicationcount	function readconstruct
function readdescendentnodesortindex	function readnodelocation
subroutine sussingasciiload	subroutine sussingtreeindicesread
subroutine sussinghdf5load	function augmentbuildtreefromnode
subroutine history_increment	subroutine history_long_integer_trim_-forward
subroutine history_trim_forward	function listindex
function listtimenext	function listtimeprevious
function intergalacticbackgroundinternalupdate	subroutine insiturate
subroutine metallicitysplitrate	subroutine spherical_collapse_matter_-lambda_nonlinear_mapping

program test_search
function intergalacticmediumstateevolveupdate
subroutine delete_
subroutine set_
function value_

interface: search_array
Description: Generic interface for array searching routines.
Code lines: 5
Contained by: module arrays_search

function: search_array_double
Description: Searches an array, $x =$ (arrayToSearch), for value, v(=valueToFind), to find the index i such that $x(i) \leq v < x(i+1)$.
Code lines: 11
Contained by: module arrays_search
Modules used: fgsl

function: search_array_for_closest
Description: Searches an array, $x =$ (arrayToSearch), for the entry closest to value, v(=valueToFind) and returns the index of that element in the array. Optionally, a tolerance may be specified within which the two values must match.
Code lines: 41
Contained by: module arrays_search
Modules used: fgsl galacticus_error numerical_comparison

function: search_array_integer8
Description: Searches a long integer array, $x =$ (arrayToSearch), for value, v(=valueToFind), to find the index i such that $x(i) \leq v < x(i+1)$.
Code lines: 44
Contained by: module arrays_search
Modules used: kind_numbers

function: search_array_varstring
Description: Searches an array, $x =$ (arrayToSearch), for value, v(=valueToFind), to find the index i such that $x(i) = v$. With this algorithm, if multiple elements of $x()$ have the same value, then the largest value of i for which $x(i) = v$ occurs will be returned.
Code lines: 46
Contained by: module arrays_search
Modules used: iso_varying_string

interface: search_indexed
Description: Generic interface for array searching routines using indexing.
Code lines: 3
Contained by: module arrays_search

function: search_indexed_integer8
Description: Searches a long integer array, $x =$ (arrayToSearch), which is rank ordered when indexed by arrayIndex, for value, v(=valueToFind), to find the index i such that $x(i) \leq v < x(i+1)$.
Code lines: 46

Contained by: module arrays_search
Modules used: iso_c_binding kind_numbers

file: numerical.sort.F90
Description: Contains a module which implements sorting sequences.
Code lines: 330

module: sort
Description: Implements sorting.
Code lines: 308
Contained by: file numerical.sort.F90
Modules used: fgsl iso_c_binding iso_varying_string
Used by:

function squareconstructorinternal	subroutine windowread
subroutine interface_camb_transfer_-function	subroutine buildconstructmasses
subroutine fixedmassconstruct	subroutine readconstruct
subroutine sampleddistributionconstruct	subroutine unionconstruct
function readconstruct	subroutine readcreatenodeindices
subroutine readrootnodeaffinitiesinitial	subroutine sussingasciiload
subroutine sussingtreeindicesread	subroutine sussinghdf5load
subroutine galacticusforestindicesread	function augmentconstructorinternal
subroutine augmentoperate	subroutine exportoperate
function regridtimesconstructorinternal	subroutine gaussianregressionfitvariogram
function multivectorizedcompositegausskronrod1devaluate	function multivectorizedcompositetrapezoidal1devaluate
subroutine stellar_luminosities_-initialize	function listconstructorparameters
subroutine differentialevolutionposterior	subroutine particleswarmposterior
subroutine latinhypercubeinitialize	subroutine evolveforestsperform
program test_sort	function mpimedianarray

function: compare_double
Description: Comparison function for double precision data.
Code lines: 16
Contained by: module sort

function: compare_integer
Description: Comparison function for integer data.
Code lines: 16
Contained by: module sort

function: compare_integer8
Description: Comparison function for integer data.
Code lines: 16
Contained by: module sort

interface: `sort_do`
Description: Generic interface to in-place sort routines.
Code lines: 7
Contained by: module `sort`

subroutine: `sort_do_double`
Description: Given an unsorted double precision `array`, sorts it in place.
Code lines: 7
Contained by: module `sort`

subroutine: `sort_do_double_both`
Description: Given an unsorted double precision `array`, sorts it in place while also rearranging `array2` in the same way.
Code lines: 19
Contained by: module `sort`
Modules used: `kind_numbers`

subroutine: `sort_do_double_c`
Description: Do a double precision sort.
Code lines: 12
Contained by: module `sort`

subroutine: `sort_do_integer`
Description: Given an unsorted integer `array`, sorts it in place.
Code lines: 7
Contained by: module `sort`

subroutine: `sort_do_integer8`
Description: Given an unsorted long integer `array`, sorts it in place.
Code lines: 8
Contained by: module `sort`
Modules used: `kind_numbers`

subroutine: `sort_do_integer8_both`
Description: Given an unsorted long integer `array`, sorts it in place while also rearraning `array2` in the same way.
Code lines: 19
Contained by: module `sort`
Modules used: `kind_numbers`

subroutine: `sort_do_integer8_c`
Description: Do a long integer sort.
Code lines: 13
Contained by: module `sort`
Modules used: `kind_numbers`

subroutine: `sort_do_integer_c`
Description: Do a integer sort.

Code lines: 12
Contained by: module sort

interface: sort_index_do
Description: Generic interface to index sort routines.
Code lines: 5
Contained by: module sort

function: sort_index_do_double
Description: Given an unsorted double array, sorts it in place.
Code lines: 10
Contained by: module sort
Modules used: kind_numbers

subroutine: sort_index_do_double_c
Description: Do an double sort.
Code lines: 15
Contained by: module sort
Modules used: kind_numbers

function: sort_index_do_integer
Description: Given an unsorted integer array, sorts it in place.
Code lines: 10
Contained by: module sort
Modules used: kind_numbers

function: sort_index_do_integer8
Description: Given an unsorted integer array, sorts it in place.
Code lines: 10
Contained by: module sort
Modules used: kind_numbers

subroutine: sort_index_do_integer8_c
Description: Do a integer sort.
Code lines: 15
Contained by: module sort
Modules used: kind_numbers

subroutine: sort_index_do_integer_c
Description: Do an integer sort.
Code lines: 15
Contained by: module sort
Modules used: kind_numbers

interface: sortbyindex
Description: Generic interface to in-place sort routines using a supplied index.
Code lines: 3
Contained by: module sort

subroutine: `sortindex`
Description: Given an `array`, sort it in place using the supplied index.
Code lines: 14
Contained by: module `sort`
Modules used: `kind_numbers`

file: `numerical.sort.topological.F90`
Description: Contains a module which implements topological sorting.
Code lines: 85

module: `sorting_topological`
Description: Implements topological sorting.
Code lines: 63
Contained by: file `numerical.sort.topological.F90`
Used by: program `test_sort_topological`

subroutine: `sort_topological`
Description: Topological sorting function. Based on the example from Rosetta Code. Arguments are:

`countObjects`→ the number of objects to be sorted;

`countDependencies`→ the number of dependencies;

`dependencies`→ an array of dependencies, such that `dependencies(:,1)` depends on `dependencies(:,2)`;

`order`← an array giving the order of the objects after sorting;

`countOrdered`← a count of the objects which were ordered by the sort, such that `order(1:countOrdered)` contains the ordered objects, while the remainder of `order()` contains objects that were unordered (i.e. had no dependencies).

The unordered objects are those for which no solution is available—i.e. the graph is not acyclic. So, if `order<countObjects` then one or more circular dependencies existed in the graph.

Code lines: 53
Contained by: module `sorting_topological`
Modules used: `galacticus_error`

file: `objects.Nbody_simulation_data.F90`
Description: Contains a module which provides a class to store N-body simulation data.
Code lines: 54

module: `nbody_simulation_data`
Description: Provides a class to store N-body simulation data.
Code lines: 32
Contained by: file `objects.Nbody_simulation_data.F90`
Modules used: `io_hdf5` `kind_numbers`
Used by: module `nbody_importers` module `nbody_operators`

subroutine nbodyanalyzeperform

interface: nbodydata
Code lines: 2
Contained by: module nbody_simulation_data

function: nbodydataconstructor
Description: A default constructor for the nBodyData class.
Code lines: 6
Contained by: module nbody_simulation_data

file: objects.abundances.F90
Description: Contains a module which defines the abundances structure used for describing elemental abundances in Galacticus.
Code lines: 1047

module: abundances_structure
Description: Defines the abundances structure used for describing elemental abundances in Galacticus.
Code lines: 1025
Contained by: file objects.abundances.F90
Modules used: iso_varying_string numerical_constants_astronomical
Used by:
module accretion_halos
function coldmodechemicalmasses
function coldmodecoldmodefraction
function simpleaccretedmassmetals
function simpleaccretionratemetals
function simplechemicalmasses
function zeroaccretedmassmetals
function zeroaccretionratemetals
program benchmark_stellar_populations_luminosities
subroutine ciefilechemicaldensities
function ciefileelectrondensity
function ciefileelectrondensitydensitylogslope
function ciefileelectrondensitytemperaturelogslope
module chemical_states
function ciefilecoolingfunction
function ciefilecoolingfunctiondensitylogslope
function ciefilecoolingfunctiontemperaturelogslope
function cmbcomptoncoolingfunction
function cmbcomptoncoolingfunctiondensitylogslope
function cmbcomptoncoolingfunctiontemperaturelogslope
module cooling_functions
function molecularhydrogengallipallacoolingfunction
function molecularhydrogengallipallacoolingfunctiondensitylogslope
function molecularhydrogengallipallacoolingfunctiontemperaturelogslope
function summationcoolingfunction
function summationcoolingfunctiondensitylogslope
function summationcoolingfunctiontemperaturelogslope
function betaprofileconstructorinternal
function betaprofileradius
function betaprofileradiusgrowthrate
function isothermalconstructorinternal
function isothermalradius

file cooling.cooling_radius.simple.F90
function simpleradiusgrowthrate
function massmetallicityandrews2013constructorinternal
function metallicity12lognhconstructorparameters
function integrandcomptiony
function lmnstyemssnlineextract
function metallicityismconstructorparameters
function metallicityismextract
module galacticus_nodes

subroutine node_component_disk_standard_post_step
subroutine node_component_disk_standard_satellite_merging
subroutine node_component_disk_very_simple_analytic_solver
subroutine node_component_disk_very_simple_post_step
subroutine node_component_disk_very_simple_rates
subroutine node_component_disk_very_simple_scale_set
subroutine node_component_hot_halo_cold_mode_initialize
subroutine node_component_hot_halo_cold_mode_outflow_return
subroutine node_component_hot_halo_cold_mode_rate_compute
subroutine node_component_hot_halo_cold_mode_scale_set
subroutine node_component_hot_halo_outflow_tracking_rate_compute
subroutine node_component_hot_halo_standard_formation
subroutine node_component_hot_halo_standard_node_merger
subroutine node_component_hot_halo_standard_outflowing_abundances_rate
subroutine node_component_hot_halo_standard_promote
subroutine node_component_hot_halo_standard_push_to_cooling_pipes

function simpleconstructorinternal
module cooling_times
function massmetallicityblanc2017constructorinternal
function sedfitevaluate

function integrandluminosityxray
function lmnstystllrchrltfll2000extract
function metallicityismdescription

function metallicityismname
subroutine node_component_disk_standard_initialize
subroutine node_component_disk_standard_rate_compute
subroutine node_component_disk_standard_scale_set
subroutine node_component_disk_very_simple_post_evolve
subroutine node_component_disk_very_simple_rate_compute
subroutine node_component_disk_very_simple_satellite_merging
subroutine node_component_hot_halo_cold_mode_formation
subroutine node_component_hot_halo_cold_mode_node_merger
subroutine node_component_hot_halo_cold_mode_push_to_cooling_pipes
subroutine node_component_hot_halo_cold_mode_satellite_merging
subroutine node_component_hot_halo_cold_mode_tree_initialize
subroutine node_component_hot_halo_outflow_tracking_scale_set
subroutine node_component_hot_halo_standard_initialize
subroutine node_component_hot_halo_standard_outflow_return
subroutine node_component_hot_halo_standard_post_evolve
subroutine node_component_hot_halo_standard_push_from_halo
subroutine node_component_hot_halo_standard_rate_compute

subroutine node_component_hot_halo_standard_satellite_merging
subroutine node_component_hot_halo_standard_scale_set
subroutine node_component_hot_halo_standard_tree_initialize
subroutine node_component_hot_halo_very_simple_node_merger
subroutine node_component_hot_halo_very_simple_outflowing_abundances_rate
subroutine node_component_hot_halo_very_simple_post_evolve
subroutine node_component_hot_halo_very_simple_push_to_cooling_pipes
subroutine node_component_hot_halo_very_simple_satellite_merging
subroutine node_component_hot_halo_very_simple_scale_set
subroutine node_component_hot_halo_very_simple_tree_initialize
subroutine node_component_hot_halo_vs_delayed_node_merger
subroutine node_component_hot_halo_vs_delayed_outflowing_abundances_rate
subroutine node_component_hot_halo_vs_delayed_post_evolve
subroutine node_component_hot_halo_vs_delayed_rate_compute
subroutine node_component_hot_halo_vs_delayed_satellite_merging
subroutine node_component_hot_halo_vs_delayed_scale_set
subroutine node_component_hot_halo_vs_delayed_tree_initialize
subroutine node_component_spheroid_standard_energy_gas_input_rate
subroutine node_component_spheroid_standard_initialize
subroutine node_component_spheroid_standard_mass_gas_sink_rate
subroutine node_component_spheroid_standard_post_step
subroutine node_component_spheroid_standard_rate_compute
subroutine node_component_spheroid_standard_satellite_merging
subroutine node_component_spheroid_standard_scale_set
subroutine node_component_spheroid_very_simple_post_step
subroutine node_component_spheroid_very_simple_rate_compute
subroutine node_component_spheroid_very_simple_rates
subroutine node_component_spheroid_very_simple_satellite_merging
subroutine node_component_spheroid_very_simple_scale_set
subroutine stellar_luminosities_set
function intergalacticbackgroundinternalupdate
module star_formation_histories
subroutine metallicitysplitrate
function blitz2006rate
function kennicuttschmidtrate
file star_formation.rate_surface_density.disks.Krumholz2009.F90
function extendedschmidtrate
module stellar_populations
module stellar_population_luminosities
module stellar_population_properties
module stellar_population_selectors
module stellar_population_spectra
function fileluminosity
program test_abundances
program test_cooling_functions
program test_stellar_populations
program test_stellar_populations_luminosities

interface: abs

Code lines: 2

Contained by: module abundances_structure

interface: `abundances`
Description: Constructors for the `abundances` class.
Code lines: 3
Contained by: module `abundances_structure`

function: `abundances_abs`
Description: Return an element-by-element `abs()` on an abundances objects.
Code lines: 9
Contained by: module `abundances_structure`

function: `abundances_add`
Description: Add two abundances objects.
Code lines: 17
Contained by: module `abundances_structure`

subroutine: `abundances_allocate_elemental_values`
Description: Ensure that the `elementalValue` array in an `abundances` is allocated.
Code lines: 8
Contained by: module `abundances_structure`
Modules used: `memory_management`

function: `abundances_atomic_index`
Description: Return the atomic index for the specified entry in the abundances structure.
Code lines: 24
Contained by: module `abundances_structure`
Modules used: `galacticus_error`

subroutine: `abundances_builder`
Description: Build a `abundances` object from the given XML `abundancesDefinition`.
Code lines: 29
Contained by: module `abundances_structure`
Modules used: `fox_dom` `galacticus_error`

subroutine: `abundances_deserialize`
Description: Pack abundances from an array into an abundances structure.
Code lines: 17
Contained by: module `abundances_structure`

subroutine: `abundances_destroy`
Description: Destroy an abundances object.
Code lines: 8
Contained by: module `abundances_structure`
Modules used: `memory_management`

function: `abundances_divide`
Description: Divide an abundances object by a scalar.
Code lines: 12
Contained by: module `abundances_structure`

subroutine: `abundances_dump`
Description: Reset an abundances object.
Code lines: 23
Contained by: module `abundances_structure`
Modules used: `galacticus_display`

subroutine: `abundances_dump_raw`
Description: Dump an abundances object to binary.
Code lines: 12
Contained by: module `abundances_structure`

function: `abundances_get_metallicity`
Description: Return the metallicity of the `self` structure.
Code lines: 36
Contained by: module `abundances_structure`
Modules used: `galacticus_error`

function: `abundances_helium_mass_fraction`
Description: Returns the mass fraction of helium.
Code lines: 10
Contained by: module `abundances_structure`

function: `abundances_helium_number_fraction`
Description: Returns the mass fraction of helium.
Code lines: 13
Contained by: module `abundances_structure`

function: `abundances_hydrogen_mass_fraction`
Description: Returns the mass fraction of hydrogen.
Code lines: 11
Contained by: module `abundances_structure`

function: `abundances_hydrogen_number_fraction`
Description: Returns the number fraction of hydrogen.
Code lines: 13
Contained by: module `abundances_structure`

subroutine: `abundances_increment`
Description: Increment an abundances object.
Code lines: 12
Contained by: module `abundances_structure`

function: `abundances_index_from_name`
Description: Return the index of an element in the elements array given its name.
Code lines: 14
Contained by: module `abundances_structure`

subroutine: `abundances_initialize`

Description: Initialize the `abundanceStructure` object module. Determines which abundances are to be tracked.
Code lines: 45
Contained by: module abundances_structure
Modules used: atomic_data input_parameters memory_management

function: `abundances_is_zero`
Description: Test whether an abundances object is zero.
Code lines: 15
Contained by: module abundances_structure

subroutine: `abundances_mass_to_mass_fraction`
Description: Convert abundance masses to mass fractions by dividing by `mass` while ensuring that the fractions remain within the range 0–1.
Code lines: 15
Contained by: module abundances_structure

subroutine: `abundances_mass_to_mass_fraction_packed`
Description: Convert abundance masses to mass fractions by dividing by `mass` while ensuring that the fractions remain within the range 0–1.
Code lines: 29
Contained by: module abundances_structure

function: `abundances_max`
Description: Return an element-by-element `max()` on two abundances objects.
Code lines: 9
Contained by: module abundances_structure

function: `abundances_multiply`
Description: Multiply an abundances object by a scalar.
Code lines: 12
Contained by: module abundances_structure

function: `abundances_multiply_switched`
Description: Multiply a scalar by an abundances object.
Code lines: 9
Contained by: module abundances_structure

function: `abundances_names`
Description: Return a name for the specified entry in the abundances structure.
Code lines: 24
Contained by: module abundances_structure
Modules used: galacticus_error

function: `abundances_non_static_size_of`
Description: Return the size of any non-static components of the object.
Code lines: 13
Contained by: module abundances_structure
Modules used: iso_c_binding

subroutine: `abundances_output`
Description: Store an abundances object in the output buffers.
Code lines: 19
Contained by: module `abundances_structure`
Modules used: `multi_counters`

subroutine: `abundances_output_count`
Description: Increment the output count to account for an abundances object.
Code lines: 10
Contained by: module `abundances_structure`

subroutine: `abundances_output_names`
Description: Assign names to output buffers for an abundances object.
Code lines: 26
Contained by: module `abundances_structure`

subroutine: `abundances_post_output`
Description: Perform post-output processing of abundances objects.
Code lines: 9
Contained by: module `abundances_structure`
Modules used: `multi_counters`

function: `abundances_property_count`
Description: Return the number of properties required to track abundances. This is equal to the number of elements tracked, `elementsCount`, plus one since we always track a total metallicity.
Code lines: 10
Contained by: module `abundances_structure`

subroutine: `abundances_read_raw`
Description: Read an abundances object from binary.
Code lines: 12
Contained by: module `abundances_structure`

subroutine: `abundances_reset`
Description: Reset an abundances object.
Code lines: 11
Contained by: module `abundances_structure`

subroutine: `abundances_serialize`
Description: Pack abundances from an array into an abundances structure.
Code lines: 20
Contained by: module `abundances_structure`

subroutine: `abundances_set_metallicity`
Description: Set the metallicity of the `self` structure to `metallicity`.
Code lines: 69
Contained by: module `abundances_structure`
Modules used: `atomic_data` `galacticus_error`

subroutine: `abundances_set_to_unity`
Description: Set an abundances object to unity.
Code lines: 13
Contained by: module `abundances_structure`

function: `abundances_subtract`
Description: Subtract two abundances objects.
Code lines: 17
Contained by: module `abundances_structure`

function: `abundancesconstructorzero`
Description: A constructor for `abundances` objects which sets all content to zero.
Code lines: 8
Contained by: module `abundances_structure`

interface: `max`
Code lines: 2
Contained by: module `abundances_structure`

interface: `operator(*)`
Code lines: 2
Contained by: module `abundances_structure`

file: `objects.chemical_abundances.F90`
Description: Contains a module which defines the structure used for describing chemical abundances in Galacticus.
Code lines: 694

module: `chemical_abundances_structure`
Description: Defines the structure used for describing chemical abundances in Galacticus.
Code lines: 672
Contained by: file `objects.chemical_abundances.F90`
Modules used: `iso_varying_string`
Used by:

module `accretion_halos`	function `coldmodeaccretedmasschemicals`
function `coldmodeaccretionratechemicals`	function `coldmodechemicalmasses`
function `coldmodecoldmodefraction`	function `simpleaccretedmasschemicals`
function `simpleaccretionratechemicals`	function `simplechemicalmasses`
function `simpleconstructorinternal`	function `zeroaccretedmasschemicals`
function `zeroaccretionratechemicals`	module `chemical_reaction_rates`
subroutine `ciefilechemicaldensities`	module `chemical_states`
function `ciefilecoolingfunction`	function `ciefilecoolingfunctiondensitylogslope`
function `ciefilecoolingfunctiontemperaturelogslope`	function `cmbcomptoncoolingfunction`
function `cmbcomptoncoolingfunctiondensitylogslope`	function `cmbcomptoncoolingfunctiontemperaturelogslope`

module cooling_functions
function molecularhydrogengallipallacoolingfunction
function molecularhydrogengallipallacoolingfunctiondensitylogslope
function molecularhydrogengallipallacoolingfunctionh2plus_electron
function molecularhydrogengallipallacoolingfunctionh2
function molecularhydrogengallipallacoolingfunctionh_h2plus
function molecularhydrogengallipallacoolingfunctiontemperaturelogslope
function summationcoolingfunction
function summationcoolingfunctiondensitylogslope
function summationcoolingfunctiontemperaturelogslope
function betaprofileconstructorinternal
function betaprofileradius
function betaprofileradiusgrowthrate
function isothermalconstructorinternal
function isothermalradius
file cooling.cooling_radius.simple.F90
function simpleconstructorinternal
function simpleradiusgrowthrate
module cooling_times
function integrandluminosityxray
module galacticus_nodes
subroutine node_component_hot_halo_outflow_tracking_scale_set
subroutine node_component_hot_halo_standard_formation
subroutine node_component_hot_halo_standard_initialize
subroutine node_component_hot_halo_standard_node_merger
subroutine node_component_hot_halo_standard_outflow_return
subroutine node_component_hot_halo_standard_promote
subroutine node_component_hot_halo_standard_push_from_halo
subroutine node_component_hot_halo_standard_rate_compute
subroutine node_component_hot_halo_standard_satellite_merging
subroutine node_component_hot_halo_standard_scale_set
subroutine node_component_hot_halo_standard_tree_initialize
program test_cooling_functions

function: `chemical_abundances_add`
Description: Add two abundances objects.
Code lines: 19
Contained by: module chemical_abundances_structure

subroutine: `chemical_abundances_allocate_values`
Description: Ensure that the `chemicalValue` array in an `chemicalsStructure` is allocated.
Code lines: 14
Contained by: module chemical_abundances_structure
Modules used: memory_management

subroutine: `chemical_abundances_deserialize`
Description: Pack abundances from an array into an abundances structure.
Code lines: 17
Contained by: module chemical_abundances_structure

function: `chemical_abundances_divide`
Description: Divide a chemical abundances object by a scalar.

Code lines: 15
Contained by: module chemical_abundances_structure

subroutine: chemical_abundances_increment
Description: Increment an abundances object.
Code lines: 10
Contained by: module chemical_abundances_structure

subroutine: chemical_abundances_initialize
Description: Initialize the chemicalAbundanceStructure object module. Determines which chemicals are to be tracked.
Code lines: 56
Contained by: module chemical_abundances_structure
Modules used: chemical_structures input_parameters memory_management

function: chemical_abundances_multiply
Description: Multiply a chemical abundances object by a scalar.
Code lines: 15
Contained by: module chemical_abundances_structure

function: chemical_abundances_multiply_switched
Description: Multiply a chemical abundances object by a scalar.
Code lines: 15
Contained by: module chemical_abundances_structure

subroutine: chemical_abundances_serialize
Description: Pack abundances from an array into an abundances structure.
Code lines: 16
Contained by: module chemical_abundances_structure

function: chemical_abundances_subtract
Description: Subtract two abundances objects.
Code lines: 19
Contained by: module chemical_abundances_structure

type: chemicalabundances
Description: The structure used for describing chemical abundances in Galacticus.
Code lines: 166
Contained by: module chemical_abundances_structure

function: chemicals_abundances
Description: Returns the abundance of a molecule in the chemical abundances structure given the moleculeIndex.
Code lines: 8
Contained by: module chemical_abundances_structure

subroutine: chemicals_abundances_destroy
Description: Destroy a chemical abundances object.
Code lines: 8

Contained by: module chemical_abundances_structure
Modules used: memory_management

function: chemicals_abundances_is_zero
Description: Test whether an chemicals object is zero.
Code lines: 10
Contained by: module chemical_abundances_structure

subroutine: chemicals_abundances_reset
Description: Resets all chemical abundances to zero.
Code lines: 14
Contained by: module chemical_abundances_structure

subroutine: chemicals_abundances_set
Description: Sets the abundance of a molecule in the chemical abundances structure given the moleculeIndex.
Code lines: 15
Contained by: module chemical_abundances_structure

subroutine: chemicals_abundances_set_to_unity
Description: Resets all chemical abundances to unity.
Code lines: 14
Contained by: module chemical_abundances_structure

subroutine: chemicals_builder
Description: Build a chemicalAbundances object from the given XML chemicalsDefinition.
Code lines: 11
Contained by: module chemical_abundances_structure
Modules used: fox_dom galacticus_error

subroutine: chemicals_dump
Description: Dump all chemical values.
Code lines: 17
Contained by: module chemical_abundances_structure
Modules used: galacticus_display

subroutine: chemicals_dump_raw
Description: Dump all chemical values in binary.
Code lines: 9
Contained by: module chemical_abundances_structure

subroutine: chemicals_enforce_positive
Description: Force all chemical values to be positive, by truncating negative values to zero.
Code lines: 11
Contained by: module chemical_abundances_structure

function: chemicals_index
Description: Returns the index of a chemical in the chemical abundances structure given the chemicalName.
Code lines: 14

Contained by: module chemical_abundances_structure

subroutine: chemicals_mass_to_number
Description: Divide all chemical species by their mass in units of the atomic mass. This converts abundances by mass into abundances by number.
Code lines: 18
Contained by: module chemical_abundances_structure

function: chemicals_names
Description: Return a name for the specified entry in the chemicals structure.
Code lines: 16
Contained by: module chemical_abundances_structure
Modules used: galacticus_error

function: chemicals_non_static_size_of
Description: Return the size of any non-static components of the object.
Code lines: 13
Contained by: module chemical_abundances_structure
Modules used: iso_c_binding

subroutine: chemicals_number_to_mass
Description: Multiply all chemical species by their mass in units of the atomic mass. This converts abundances by number into abundances by mass.
Code lines: 18
Contained by: module chemical_abundances_structure

function: chemicals_property_count
Description: Return the number of properties required to track chemicals. This is equal to the number of chemicals tracked, chemicalsCount.
Code lines: 10
Contained by: module chemical_abundances_structure

subroutine: chemicals_read_raw
Description: Read all chemical values in binary.
Code lines: 14
Contained by: module chemical_abundances_structure
Modules used: memory_management

interface: operator(*)
Code lines: 2
Contained by: module chemical_abundances_structure

file: objects.chemical_structure.F90
Description: Contains a module which implements structures that describe chemicals.
Code lines: 302

module: chemical_structures
Description: Implements structures that describe chemicals.
Code lines: 280
Contained by: file objects.chemical_structure.F90

Modules used: iso_varying_string numerical_constants_atomic numerical_constants_physical
Used by: subroutine chemical_abundances_-initialize

type: atomicbond
Description: A type that defines an atomic bond within a chemical.
Code lines: 3
Contained by: module chemical_structures

type: atomicstructure
Description: A type that defines an atom within a chemical.
Code lines: 5
Contained by: module chemical_structures

subroutine: chemical_database_get
Description: Find a chemical in the database and return it.
Code lines: 11
Contained by: module chemical_structures

function: chemical_database_get_index
Description: Find a chemical in the database and return it.
Code lines: 20
Contained by: module chemical_structures
Modules used: galacticus_error

function: chemical_structure_charge
Description: Return the charge on a chemical.
Code lines: 7
Contained by: module chemical_structures

subroutine: chemical_structure_export
Description: Export a chemical structure to a chemical markup language (CML) file.
Code lines: 70
Contained by: module chemical_structures
Modules used: fox_wxml

subroutine: chemical_structure_initialize
Description: Initialize the chemical structure database by reading the atomic structure database. Note: this implementation is not fully compatible with chemical markup language (CML), but only a limited subset of it.
Code lines: 78
Contained by: module chemical_structures
Modules used: fox_dom galacticus_error galacticus_paths

function: chemical_structure_mass
Description: Return the mass of a chemical.
Code lines: 7

Contained by: module chemical_structures

type: chemicalstructure
Description: A type that defines a chemical.
Code lines: 41
Contained by: module chemical_structures

file: objects.coordinates.F90
Description: Contains a module which implements the coordinates class.
Code lines: 559

module: coordinates
Description: Implements the coordinates class.
Code lines: 537
Contained by: file objects.coordinates.F90
Used by:

function betaprofiledensity	function betaprofiledensitylogslope
module mass_distributions	function miyamotonagaidensity
function integrandr	function integrandz
function miyamotonagaipotential	function miyamotonagairotationcurve
function miyamotonagairotationcurvegradient	function miyamotonagaisurfacedensity
function integrandsurfacedensity	function exponentialdiskdensity
function exponentialdiskpotential	function exponentialdiskrotationcurve
function exponentialdiskrotationcurvegradient	function exponentialdisksurfacedensity
function sphericalmassenclosedbysphereintegrand	function hernquistdensity
function hernquistpotential	function nfwdensity
function sersicdensity	function sersicpotential
function betaprofilepotential	subroutine particulateoperate
function kepler_orbits_position	function kepler_orbits_velocity
function node_component_disk_standard_-density	function node_component_disk_standard_-potential
function node_component_disk_standard_-surface_density	function node_component_spheroid_-standard_density
function node_component_spheroid_-standard_potential	function node_component_hot_halo_cold_-mode_density_task
subroutine node_component_satellite_-orbiting_virial_orbit_set	function orbital_angular_momentum
function spincorrelatedorbit	function randomisotropicposition
function randomisotropicvelocity	program test_mass_distributions

interface: assignment(=)
Code lines: 4
Contained by: module coordinates

type: coordinate
Description: The base coordinate object class.
Code lines: 28

Contained by: module coordinates

type: `coordinatecartesian`
Description: A Cartesian coordinate object class.
Code lines: 50
Contained by: module coordinates

type: `coordinatecylindrical`
Description: A cylindrical coordinate object class.
Code lines: 50
Contained by: module coordinates

subroutine: `coordinates_assign`
Description: Assign one coordinate object to another, automatically handling the conversion between coordinate systems.
Code lines: 16
Contained by: module coordinates

subroutine: `coordinates_assign_from`
Description: Return a 3-component vector from a `coordinate` object.
Code lines: 8
Contained by: module coordinates

subroutine: `coordinates_assign_to`
Description: Assign a 3-component vector to a `coordinate` object.
Code lines: 8
Contained by: module coordinates

subroutine: `coordinates_cartesian_from_cartesian`
Description: Create a Cartesian `coordinate` object from a Cartesian vector.
Code lines: 8
Contained by: module coordinates

subroutine: `coordinates_cartesian_set_x`
Description: Return the x-component of a Cartesian `coordinate` object.
Code lines: 8
Contained by: module coordinates

subroutine: `coordinates_cartesian_set_y`
Description: Return the y-component of a Cartesian `coordinate` object.
Code lines: 8
Contained by: module coordinates

subroutine: `coordinates_cartesian_set_z`
Description: Return the z-component of a Cartesian `coordinate` object.
Code lines: 8
Contained by: module coordinates

function: `coordinates_cartesian_to_cartesian`

Description: Return a Cartesian vector from a Cartesian `coordinate` object.
Code lines: 8
Contained by: module `coordinates`

function: `coordinates_cartesian_x`
Description: Return the x-component of a Cartesian `coordinate` object.
Code lines: 7
Contained by: module `coordinates`

function: `coordinates_cartesian_y`
Description: Return the y-component of a Cartesian `coordinate` object.
Code lines: 7
Contained by: module `coordinates`

function: `coordinates_cartesian_z`
Description: Return the z-component of a Cartesian `coordinate` object.
Code lines: 7
Contained by: module `coordinates`

subroutine: `coordinates_cylindrical_from_cartesian`
Description: Create a cylindrical `coordinate` object from a Cartesian vector.
Code lines: 12
Contained by: module `coordinates`

function: `coordinates_cylindrical_phi`
Description: Return the ϕ-component of a Cylindrical `coordinate` object.
Code lines: 7
Contained by: module `coordinates`

function: `coordinates_cylindrical_r`
Description: Return the r-component of a Cylindrical `coordinate` object.
Code lines: 7
Contained by: module `coordinates`

subroutine: `coordinates_cylindrical_set_phi`
Description: Return the ϕ-component of a Cylindrical `coordinate` object.
Code lines: 8
Contained by: module `coordinates`

subroutine: `coordinates_cylindrical_set_r`
Description: Return the r-component of a Cylindrical `coordinate` object.
Code lines: 8
Contained by: module `coordinates`

subroutine: `coordinates_cylindrical_set_z`
Description: Return the z-component of a Cylindrical `coordinate` object.
Code lines: 8
Contained by: module `coordinates`

function: `coordinates_cylindrical_to_cartesian`
Description: Return a Cartesian vector from a cylindrical `coordinate` object.
Code lines: 12
Contained by: module `coordinates`

function: `coordinates_cylindrical_z`
Description: Return the z-component of a Cylindrical `coordinate` object.
Code lines: 7
Contained by: module `coordinates`

subroutine: `coordinates_null_from`
Description: Set generic coordinate object from Cartesian point. Simply quits with an error.
Code lines: 10
Contained by: module `coordinates`
Modules used: `galacticus_error`

function: `coordinates_null_to`
Description: Convert generic coordinate object to Cartesian point. Simply quits with an error.
Code lines: 11
Contained by: module `coordinates`
Modules used: `galacticus_error`

function: `coordinates_radius_cylindrical`
Code lines: 15
Contained by: module `coordinates`

subroutine: `coordinates_spherical_from_cartesian`
Description: Create a spherical `coordinate` object from a Cartesian vector.
Code lines: 18
Contained by: module `coordinates`

function: `coordinates_spherical_phi`
Description: Return the ϕ-component of a Spherical `coordinate` object.
Code lines: 7
Contained by: module `coordinates`

function: `coordinates_spherical_r`
Description: Return the r-component of a Spherical `coordinate` object.
Code lines: 7
Contained by: module `coordinates`

subroutine: `coordinates_spherical_set_phi`
Description: Return the ϕ-component of a Spherical `coordinate` object.
Code lines: 8
Contained by: module `coordinates`

subroutine: `coordinates_spherical_set_r`
Description: Return the r-component of a Spherical `coordinate` object.

Code lines: 8
Contained by: module coordinates

subroutine: coordinates_spherical_set_theta
Description: Return the θ-component of a Spherical coordinate object.
Code lines: 8
Contained by: module coordinates

function: coordinates_spherical_theta
Description: Return the θ-component of a Spherical coordinate object.
Code lines: 7
Contained by: module coordinates

function: coordinates_spherical_to_cartesian
Description: Return a Cartesian vector from a spherical coordinate object.
Code lines: 12
Contained by: module coordinates

type: coordinatespherical
Description: A spherical coordinate object class.
Code lines: 50
Contained by: module coordinates

file: objects.function_class.F90
Description: Contains a module which defines the base class for all functionClass classes.
Code lines: 172

module: function_classes
Description: Defines the base class for all functionClass classes.
Code lines: 150
Contained by: file objects.function_class.F90
Modules used: iso_varying_string
Used by: module dark_matter_profiles_generic module input_parameters

function: debugstackget
Description: Get the current location from the debug stack.
Code lines: 11
Contained by: module function_classes

subroutine: debugstackpop
Description: Pop a location off the debug stack.
Code lines: 8
Contained by: module function_classes
Modules used: galacticus_error

interface: debugstackpush
Code lines: 3
Contained by: module function_classes

subroutine: `debugstackpushloc`
Description: Push a numeric-location onto the debug location stack.
Code lines: 13
Contained by: module `function_classes`
Modules used: `galacticus_error` `iso_c_binding`

subroutine: `debugstackpushstr`
Description: Push a text-location onto the debug location stack.
Code lines: 10
Contained by: module `function_classes`
Modules used: `galacticus_error`

type: `functionclass`
Description: The base class for all `functionClass` classes.
Code lines: 36
Contained by: module `function_classes`

function: `functionclassisdefault`
Description: Return true if this is the default object of this class.
Code lines: 7
Contained by: module `function_classes`

function: `functionclassreferencecountdecrement`
Description: Decrement the reference count to this object and return the new count.
Code lines: 8
Contained by: module `function_classes`

subroutine: `functionclassreferencecountincrement`
Description: Increment the reference count to this object.
Code lines: 7
Contained by: module `function_classes`

subroutine: `functionclassreferencecountreset`
Description: Reset the reference count to this object to 0.
Code lines: 7
Contained by: module `function_classes`

file: `objects.history.F90`
Description: Contains a module defining the history object type.
Code lines: 1534

module: `histories`
Description: Defines the history object type.
Code lines: 1512
Contained by: file `objects.history.F90`
Modules used: `kind_numbers`
Used by: subroutine subroutine `prunelightconeoperate` `readbuildsubhalomasshistories`

function satellitestatusextract
function satellitepresetmergeboundmass
subroutine node_component_disk_-standard_create
subroutine node_component_disk_-standard_rate_compute
subroutine node_component_disk_-standard_scale_set
subroutine node_component_disk_very_-simple_analytic_solver
subroutine node_component_disk_very_-simple_post_evolve
subroutine node_component_disk_very_-simple_rates
subroutine node_component_position_-preset_inter_tree_insert
subroutine node_component_satellite_-preset_inter_tree_insert
module node_component_spheroid_-standard
subroutine node_component_spheroid_-standard_post_evolve
subroutine node_component_spheroid_-standard_scale_set
subroutine node_component_spheroid_-very_simple_create
subroutine node_component_spheroid_-very_simple_rate_compute
subroutine node_component_spheroid_-very_simple_satellite_merging
module star_formation_histories
module galacticus_nodes
function satellitepresetnodeindex
subroutine node_component_disk_-standard_post_evolve
subroutine node_component_disk_-standard_satellite_merging
subroutine node_component_disk_-standard_star_formation_history_-output
subroutine node_component_disk_very_-simple_create
subroutine node_component_disk_very_-simple_rate_compute
subroutine node_component_disk_very_-simple_scale_set
subroutine node_component_satellite_-preset_inter_tree_attach
subroutine node_component_satellite_-preset_rate_compute
subroutine node_component_spheroid_-standard_initializor
subroutine node_component_spheroid_-standard_rate_compute
subroutine node_component_spheroid_-standard_star_formation_history_-output
subroutine node_component_spheroid_-very_simple_post_evolve
subroutine node_component_spheroid_-very_simple_rates
subroutine node_component_spheroid_-very_simple_scale_set
module stellar_population_properties

type: `history`
Description: The history object type.
Code lines: 197
Contained by: module histories

function: `history_add`
Description: Add two history objects.
Code lines: 22
Contained by: module histories
Modules used: galacticus_error

subroutine: `history_append_epoch`
Description: Append a history to a long integer history.
Code lines: 34

Contained by: module histories
Modules used: galacticus_error memory_management

subroutine: history_append_history
Description: Append a history to a long integer history.
Code lines: 31
Contained by: module histories
Modules used: galacticus_error memory_management

subroutine: history_builder
Description: Build a history object from the given XML historyDefinition.
Code lines: 11
Contained by: module histories
Modules used: fox_dom galacticus_error

subroutine: history_clone
Description: Clone a history object.
Code lines: 19
Contained by: module histories
Modules used: memory_management

subroutine: history_create
Description: Create a history object.
Code lines: 36
Contained by: module histories
Modules used: galacticus_error memory_management numerical_ranges

subroutine: history_deserialize
Description: Pack history from an array into a history structure.
Code lines: 14
Contained by: module histories
Modules used: galacticus_error

subroutine: history_destroy
Description: Destroy a history.
Code lines: 19
Contained by: module histories
Modules used: memory_management

function: history_divide
Description: Divides history data by a double precision divisor.
Code lines: 13
Contained by: module histories

subroutine: history_dump
Description: Dumps a history object.
Code lines: 24
Contained by: module histories

Modules used: galacticus_display iso_varying_string

subroutine: history_dump_raw
Description: Dumps a history object in binary.
Code lines: 14
Contained by: module histories

function: history_exists
Description: Returns true if the history has been created.
Code lines: 7
Contained by: module histories

subroutine: history_extend
Description: Extends a history to encompass the given time range.
Code lines: 123
Contained by: module histories
Modules used: galacticus_error iso_varying_string
numerical_ranges range
string_handling

subroutine: history_increment
Description: Combines the data in addHistory with that in thisHistory. This function is designed for histories that track integrated quantities (such as total mass of stars formed in a time interval for example). thisHistory will be extended if necessary to span the range of addHistory. Then, the data from addHistory will be added to that in thisHistory by finding the fraction of each timestep in addHistory that overlaps with each timestep in thisHistory and assuming that the corresponding fraction of the data value should be added to thisHistory.
Code lines: 85
Contained by: module histories
Modules used: arrays_search galacticus_error
iso_c_binding numerical_ranges

subroutine: history_interpolated_increment
Description: Adds the data in addHistory to that in thisHistory. This function is designed for histories that track instantaneous rates. The rates in addHistory are interpolated to the times in thisHistory and added to the rates in thisHistory.
Code lines: 59
Contained by: module histories
Modules used: galacticus_error iso_c_binding
numerical_interpolation

function: history_is_zero
Description: Test whether a history object is all zero.
Code lines: 10
Contained by: module histories

subroutine: history_long_integer_append_epoch
Description: Append a history to a long integer history.

Code lines: 34
Contained by: module histories
Modules used: galacticus_error memory_management

subroutine: history_long_integer_append_history
Description: Append a history to a long integer history.
Code lines: 31
Contained by: module histories
Modules used: galacticus_error memory_management

subroutine: history_long_integer_builder
Description: Build a longIntegerHistory object from the given XML historyDefinition.
Code lines: 11
Contained by: module histories
Modules used: fox_dom galacticus_error

subroutine: history_long_integer_clone
Description: Clone a longIntegerHistory object.
Code lines: 19
Contained by: module histories
Modules used: memory_management

subroutine: history_long_integer_create
Description: Create a history object.
Code lines: 36
Contained by: module histories
Modules used: galacticus_error memory_management numerical_ranges

subroutine: history_long_integer_destroy
Description: Destroy a history.
Code lines: 19
Contained by: module histories
Modules used: memory_management

subroutine: history_long_integer_dump
Description: Dumps a history object.
Code lines: 24
Contained by: module histories
Modules used: galacticus_display iso_varying_string

subroutine: history_long_integer_dump_raw
Description: Dumps a history object in binary.
Code lines: 14
Contained by: module histories

function: history_long_integer_exists
Description: Returns true if the history has been created.
Code lines: 7

Contained by: module histories

function: history_long_integer_non_static_size_of
Description: Return the size of any non-static components of the object.
Code lines: 13
Contained by: module histories
Modules used: iso_c_binding

subroutine: history_long_integer_read_raw
Description: Read a history object in binary.
Code lines: 19
Contained by: module histories
Modules used: memory_management

subroutine: history_long_integer_reset
Description: Reset a history by zeroing all elements, but leaving the structure (and times) intact.
Code lines: 7
Contained by: module histories

subroutine: history_long_integer_trim
Description: Removes outdated information from "future histories" (i.e. histories that store data for future reference). Removes all but one entry prior to the given currentTime (this allows for interpolation of the history to the current time). Optionally, the remove is done only if it will remove more than minimumPointsToRemove entries (since the removal can be slow this allows for some optimization).
Code lines: 61
Contained by: module histories
Modules used: galacticus_error iso_c_binding memory_management

subroutine: history_long_integer_trim_forward
Description: Removes all points in a history after the given time. Optionally, the removed history can be returned as removedHistory.
Code lines: 44
Contained by: module histories
Modules used: arrays_search iso_c_binding memory_management

function: history_multiply
Description: Multiplies history data by a double precision multiplier.
Code lines: 13
Contained by: module histories

function: history_multiply_switched
Description: Multiply a scalar by an history object.
Code lines: 9
Contained by: module histories

function: history_non_static_size_of

Description: Return the size of any non-static components of the object.
Code lines: 13
Contained by: module histories
Modules used: iso_c_binding

subroutine: history_read_raw
Description: Read a history object in binary.
Code lines: 19
Contained by: module histories
Modules used: memory_management

subroutine: history_reset
Description: Reset a history by zeroing all elements, but leaving the structure (and times) intact.
Code lines: 7
Contained by: module histories

subroutine: history_serialize
Description: Pack history from an array into an history structure.
Code lines: 9
Contained by: module histories

function: history_serialize_count
Description: Return the number of properties required to track a history.
Code lines: 11
Contained by: module histories

subroutine: history_set_to_unity
Description: Reset a history by zeroing all elements, but leaving the structure (and times) intact.
Code lines: 7
Contained by: module histories

function: history_subtract
Description: Subtract two history objects.
Code lines: 24
Contained by: module histories
Modules used: galacticus_error

subroutine: history_timesteps
Description: Return an array of time intervals in thisHistory.
Code lines: 26
Contained by: module histories
Modules used: memory_management numerical_ranges

subroutine: history_trim

Description: Removes outdated information from "future histories" (i.e. histories that store data for future reference). Removes all but one entry prior to the given `currentTime` (this allows for interpolation of the history to the current time). Optionally, the remove is done only if it will remove more than `minimumPointsToRemove` entries (since the removal can be slow this allows for some optimization).

Code lines: 61

Contained by: module `histories`

Modules used: `galacticus_error` `iso_c_binding`

memory_management

subroutine: `history_trim_forward`

Description: Removes all points in a history after the given `time`. Optionally, the removed history can be returned as `removedHistory`.
Code lines: 44
Contained by: module histories
Modules used: arrays_search, iso_c_binding, memory_management

type: `longintegerhistory`

Description: The history object type.
Code lines: 102
Contained by: module histories

interface: `operator(*)`

Code lines: 2
Contained by: module histories

file: `objects.kepler_orbits.F90`

Description: Contains a module which defines an orbit structure for use in Galacticus.
Code lines: 1169

module: `kepler_orbits`

Description: Defines an orbit structure for use in Galacticus.
Code lines: 1147
Contained by: file objects.kepler_orbits.F90
Used by: function font2008force; function readorbitconstruct; subroutine readtimeuntilmergingsubresolution; function satelliteorbitalextremaextract; subroutine node_component_dynamics_statistics_bars_record; module node_component_satellite_standard; module node_component_spin_vitvitska; module satellite_merging_timescales; function villalobos2013timeuntilmerging; function presettimeuntilmerging; module virial_orbits; function satellite_orbit_equivalent_circular_orbit_radius; subroutine readintertreemergetimeset; subroutine readscanformergers; subroutine assignorbitsoperate; module galacticus_nodes; module node_component_satellite_orbiting; subroutine node_component_satellite_very_simple_create; function boylankolchin2008timeuntilmerging; function laceycole1993timeuntilmerging; function wetzelwhite2010timeuntilmerging; function randomtimeuntilmerging; function satellite_orbit_convert_to_current_potential; subroutine satellite_orbit_extremum_phase_space_coordinates

function sphericalsymmetrytidaltensorradial

program tests_kepler_orbits

function: `kepler_orbits_angular_momentum`
Description: Return the angular momentum for this orbit.
Code lines: 11
Contained by: module kepler_orbits

subroutine: `kepler_orbits_angular_momentum_set`
Description: Sets the tangential velocity to the specified value.
Code lines: 10
Contained by: module kepler_orbits

function: `kepler_orbits_apocenter_radius`
Description: Return the apocenter radius for this orbit.
Code lines: 18
Contained by: module kepler_orbits

subroutine: `kepler_orbits_apocenter_radius_set`
Description: Sets the apocenter radius to the specified value.
Code lines: 10
Contained by: module kepler_orbits

subroutine: `kepler_orbits_assert_is_defined`
Description: Assert that an orbit is defined - quit with an error if it is not.
Code lines: 8
Contained by: module kepler_orbits
Modules used: galacticus_error

subroutine: `kepler_orbits_builder`
Description: Build a `keplerOrbit` object from the given XML `keplerOrbitDefinition`.
Code lines: 43
Contained by: module kepler_orbits
Modules used: fox_dom galacticus_error

subroutine: `kepler_orbits_destroy`
Description: Destroy an orbit.
Code lines: 8
Contained by: module kepler_orbits

subroutine: `kepler_orbits_dump`
Description: Reset an orbit to a null state.
Code lines: 63
Contained by: module kepler_orbits
Modules used: galacticus_display iso_varying_string

subroutine: `kepler_orbits_dump_raw`
Description: Dump a `keplerOrbit` object in binary.
Code lines: 18

Contained by: module kepler_orbits

function: kepler_orbits_eccentricity
Description: Return the eccentricity for this orbit.
Code lines: 14
Contained by: module kepler_orbits

subroutine: kepler_orbits_eccentricity_set
Description: Sets the tangential velocity to the specified value.
Code lines: 10
Contained by: module kepler_orbits

function: kepler_orbits_energy
Description: Return the energy for this orbit.
Code lines: 16
Contained by: module kepler_orbits
Modules used: numerical_constants_physical

subroutine: kepler_orbits_energy_set
Description: Sets the tangential velocity to the specified value.
Code lines: 10
Contained by: module kepler_orbits

function: kepler_orbits_epsilon
Description: Return the angle ϵ for this orbit.
Code lines: 9
Contained by: module kepler_orbits
Modules used: galacticus_error

subroutine: kepler_orbits_epsilon_set
Description: Sets the ϵ to the specified value.
Code lines: 10
Contained by: module kepler_orbits

function: kepler_orbits_host_mass
Description: Return the host mass for this orbit.
Code lines: 9
Contained by: module kepler_orbits
Modules used: galacticus_error

function: kepler_orbits_is_bound
Description: Returns true if the orbit is bound.
Code lines: 10
Contained by: module kepler_orbits

function: kepler_orbits_is_defined

Description: Returns true if the orbit is fully defined. For the orbits consider here, in which we don't care about the orientation of the orbital plane or the argument of pericenter, this requires that three orbital parameter be set (in addition to the masses of the orbitting bodies).
Code lines: 27
Contained by: module `kepler_orbits`

subroutine: `kepler_orbits_masses_set`
Description: Sets the masses of the two orbitting objects in a `keplerOrbit` object.
Code lines: 11
Contained by: module `kepler_orbits`

function: `kepler_orbits_non_static_size_of`
Description: Return the size of any non-static components of the object.
Code lines: 10
Contained by: module `kepler_orbits`
Modules used: `iso_c_binding`

subroutine: `kepler_orbits_output`
Description: Store a `keplerOrbit` object in the output buffers.
Code lines: 23
Contained by: module `kepler_orbits`
Modules used: `kind_numbers` `multi_counters`

subroutine: `kepler_orbits_output_count`
Description: Increment the output count to account for a `keplerOrbit` object.
Code lines: 10
Contained by: module `kepler_orbits`

subroutine: `kepler_orbits_output_names`
Description: Assign names to output buffers for a `keplerOrbit` object.
Code lines: 30
Contained by: module `kepler_orbits`
Modules used: `numerical_constants_astronomical`

function: `kepler_orbits_pericenter_radius`
Description: Return the pericenter radius for this orbit.
Code lines: 14
Contained by: module `kepler_orbits`

subroutine: `kepler_orbits_pericenter_radius_set`
Description: Sets the pericenter radius to the specified value.
Code lines: 10
Contained by: module `kepler_orbits`

function: `kepler_orbits_phi`
Description: Return the angle ϕ for this orbit.
Code lines: 9
Contained by: module `kepler_orbits`
Modules used: `galacticus_error`

subroutine: `kepler_orbits_phi_set`
Description: Sets the angle ϕ to the specified value.
Code lines: 10
Contained by: module `kepler_orbits`

function: `kepler_orbits_position`
Description: Return the position of the orbit in Cartesian coordinates.
Code lines: 9
Contained by: module `kepler_orbits`
Modules used: `coordinates`

subroutine: `kepler_orbits_post_output`
Description: Perform post-output processing of a `keplerOrbit` object.
Code lines: 8
Contained by: module `kepler_orbits`

subroutine: `kepler_orbits_propagate`
Description: Propagate an orbit along its path.
Code lines: 33
Contained by: module `kepler_orbits`
Modules used: `galacticus_error` `numerical_constants_physical`

function: `kepler_orbits_radius`
Description: Return the radius for this orbit.
Code lines: 9
Contained by: module `kepler_orbits`
Modules used: `galacticus_error`

subroutine: `kepler_orbits_radius_set`
Description: Sets the radius to the specified value.
Code lines: 10
Contained by: module `kepler_orbits`

subroutine: `kepler_orbits_read_raw`
Description: Read a `keplerOrbit` object in binary.
Code lines: 18
Contained by: module `kepler_orbits`

subroutine: `kepler_orbits_reset`
Description: Reset an orbit to a null state.
Code lines: 17
Contained by: module `kepler_orbits`

function: `kepler_orbits_semi_major_axis`
Description: Return the semi-major axis for this orbit.
Code lines: 14
Contained by: module `kepler_orbits`

subroutine: `kepler_orbits_semi_major_axis_set`
Description: Sets the semi-major axis to the specified value.
Code lines: 10
Contained by: module `kepler_orbits`

function: `kepler_orbits_specific_reduced_mass`
Description: Return the specific reduced mass for this orbit.
Code lines: 9
Contained by: module `kepler_orbits`
Modules used: `galacticus_error`

function: `kepler_orbits_theta`
Description: Return the angle θ for this orbit.
Code lines: 9
Contained by: module `kepler_orbits`
Modules used: `galacticus_error`

subroutine: `kepler_orbits_theta_set`
Description: Sets the angle θ to the specified value.
Code lines: 10
Contained by: module `kepler_orbits`

function: `kepler_orbits_velocity`
Description: Return the position of the orbit in Cartesian coordinates.
Code lines: 18
Contained by: module `kepler_orbits`
Modules used: `coordinates` `vectors`

function: `kepler_orbits_velocity_radial`
Description: Return the radial velocity for this orbit.
Code lines: 19
Contained by: module `kepler_orbits`
Modules used: `galacticus_error`

subroutine: `kepler_orbits_velocity_radial_set`
Description: Sets the radial velocity to the specified value.
Code lines: 10
Contained by: module `kepler_orbits`

function: `kepler_orbits_velocity_scale`
Description: Return the velocity scale for the orbit.
Code lines: 12
Contained by: module `kepler_orbits`
Modules used: `galacticus_error` `numerical_constants_physical`

function: `kepler_orbits_velocity_tangential`
Description: Return the tangential velocity for this orbit.
Code lines: 24
Contained by: module `kepler_orbits`
Modules used: `galacticus_error`

subroutine: `kepler_orbits_velocity_tangential_set`

Description: Sets the tangential velocity to the specified value.
Code lines: 10
Contained by: module kepler_orbits

interface: keplerorbit
Description: Constructors for Kepler orbits.
Code lines: 3
Contained by: module kepler_orbits

function: keplerorbitconstructornull
Description: Null constructor for Kepler orbit objects.
Code lines: 6
Contained by: module kepler_orbits

file: objects.merger_tree_data.F90
Description: Contains a module which implements an object to store merger tree data for processing into Galacticus's preferred file format.
Code lines: 2116

module: merger_tree_data_structure
Description: Implements an object to store merger tree data for processing into Galacticus's preferred file format.
Code lines: 2092
Contained by: file objects.merger_tree_data.F90
Modules used: iso_c_binding, iso_varying_string, kind_numbers
Used by: function exportconstructorinternal; function exportconstructorparameters; subroutine exportoperate; function mergertreefilebuilderconstructorparameters; subroutine mergertreefilebuilderperform

subroutine: merger_tree_data_construct_particle_indices
Description: If we have most-bound particle indices and particle data has been read, construct arrays giving position of particle data for each node.
Code lines: 40
Contained by: module merger_tree_data_structure
Modules used: galacticus_error, memory_management

subroutine: merger_tree_data_set_subhalo_masses
Description: Set the masses of any subhalos (which have zero mass by default) based on particle count.
Code lines: 12
Contained by: module merger_tree_data_structure
Modules used: galacticus_error

subroutine: merger_tree_data_structure_add_metadata
Description: Add a metadatum.
Code lines: 57
Contained by: module merger_tree_data_structure
Modules used: galacticus_error, memory_management

subroutine: `merger_tree_data_structure_add_metadata_double`
Description: Add a double metadatum.
Code lines: 12
Contained by: module `merger_tree_data_structure`

subroutine: `merger_tree_data_structure_add_metadata_integer`
Description: Add an integer metadatum.
Code lines: 10
Contained by: module `merger_tree_data_structure`

subroutine: `merger_tree_data_structure_add_metadata_text`
Description: Add a double metadatum.
Code lines: 10
Contained by: module `merger_tree_data_structure`

subroutine: `merger_tree_data_structure_convert_property_units`
Description: Convert the property with inconsistent units.
Code lines: 54
Contained by: module `merger_tree_data_structure`
Modules used: `galacticus_error`

subroutine: `merger_tree_data_structure_export`
Description: Output a set of merger trees to an HDF5 file.
Code lines: 27
Contained by: module `merger_tree_data_structure`
Modules used: `galacticus_error` `hdf5` `string_handling`

subroutine: `merger_tree_data_structure_export_galacticus`
Description: Output a set of merger trees to a Galacticus-format HDF5 file.
Code lines: 266
Contained by: module `merger_tree_data_structure`
Modules used: `file_utilities` `galacticus_error` `hdf5` `io_hdf5` `iso_c_binding` `memory_management` `string_handling`

subroutine: `merger_tree_data_structure_export_irate`
Description: Output a set of merger trees to an IRATE-format HDF5 file.
Code lines: 283
Contained by: module `merger_tree_data_structure`
Modules used: `array_utilities` `file_utilities` `galacticus_error` `hdf5` `io_hdf5` `memory_management`

subroutine: `merger_tree_data_structure_make_references`
Description: Specify whether or not to make merger tree dataset references.
Code lines: 8

Contained by: module merger_tree_data_structure

subroutine: merger_tree_data_structure_read_ascii
Description: Read in merger tree data from an ASCII file.
Code lines: 344
Contained by: module merger_tree_data_structure
Modules used: file_utilities, galacticus_display, galacticus_error, memory_management, string_handling

subroutine: merger_tree_data_structure_read_particles_ascii
Description: Read in particle data from an ASCII file.
Code lines: 125
Contained by: module merger_tree_data_structure
Modules used: file_utilities, galacticus_error, memory_management, string_handling

subroutine: merger_tree_data_structure_reset
Description: Reset a merger tree data object.
Code lines: 68
Contained by: module merger_tree_data_structure
Modules used: memory_management

subroutine: merger_tree_data_structure_set_conversion_factor
Description: Set Conversion factor for property type with inconsistent unit.
Code lines: 14
Contained by: module merger_tree_data_structure
Modules used: galacticus_error

subroutine: merger_tree_data_structure_set_forest_count
Description: Set the total number of trees in merger trees.
Code lines: 10
Contained by: module merger_tree_data_structure

subroutine: merger_tree_data_structure_set_includes_hubble_flow
Description: Set the particle mass used in the trees.
Code lines: 10
Contained by: module merger_tree_data_structure

subroutine: merger_tree_data_structure_set_includes_subhalo_masses
Description: Set the particle mass used in the trees.
Code lines: 10
Contained by: module merger_tree_data_structure

subroutine: merger_tree_data_structure_set_is_periodic
Description: Set whether or not positions are periodic.
Code lines: 10
Contained by: module merger_tree_data_structure

subroutine: `merger_tree_data_structure_set_node_count`
Description: Set the total number of nodes in merger trees.
Code lines: 10
Contained by: module `merger_tree_data_structure`

subroutine: `merger_tree_data_structure_set_particle_count`
Description: Set the total number of particles in merger trees.
Code lines: 10
Contained by: module `merger_tree_data_structure`

subroutine: `merger_tree_data_structure_set_particle_mass`
Description: Set the particle mass used in the trees.
Code lines: 10
Contained by: module `merger_tree_data_structure`

subroutine: `merger_tree_data_structure_set_particle_property_column`
Description: Set column mapping from the input file.
Code lines: 24
Contained by: module `merger_tree_data_structure`
Modules used: `memory_management`

subroutine: `merger_tree_data_structure_set_property_column`
Description: Set column mapping from the input file.
Code lines: 24
Contained by: module `merger_tree_data_structure`
Modules used: `memory_management`

subroutine: `merger_tree_data_structure_set_property_double`
Description: Set a property in the merger trees.
Code lines: 73
Contained by: module `merger_tree_data_structure`
Modules used: `galacticus_error` `memory_management`

subroutine: `merger_tree_data_structure_set_property_integer8`
Description: Set a property in the merger trees.
Code lines: 44
Contained by: module `merger_tree_data_structure`
Modules used: `galacticus_error` `memory_management`

subroutine: `merger_tree_data_structure_set_self_contained`
Description: Set the particle mass used in the trees.
Code lines: 10
Contained by: module `merger_tree_data_structure`

subroutine: `merger_tree_data_structure_set_self_hosting_halo_id`
Description: Set the host ID in case of self-hosting halos. Default is host ID = node ID.
Code lines: 10
Contained by: module `merger_tree_data_structure`

subroutine: `merger_tree_data_structure_set_tree_indices`
Description: Set the merger tree index arrays.
Code lines: 41
Contained by: module `merger_tree_data_structure`
Modules used: `memory_management`

subroutine: `merger_tree_data_structure_set_units`
Description: Set the units system.
Code lines: 43
Contained by: module `merger_tree_data_structure`
Modules used: `galacticus_error`

subroutine: `merger_tree_data_validate_trees`
Description: Validate the merger trees.
Code lines: 11
Contained by: module `merger_tree_data_structure`
Modules used: `galacticus_error`

type: `mergertreedata`
Description: A structure that holds raw merger tree data.
Code lines: 166
Contained by: module `merger_tree_data_structure`

subroutine: `store_unit_attributes_galacticus`
Description: Store attributes describing the unit system.
Code lines: 13
Contained by: module `merger_tree_data_structure`
Modules used: `io_hdf5`

subroutine: `store_unit_attributes_irate`
Description: Store unit attributes in IRATE format files.
Code lines: 39
Contained by: module `merger_tree_data_structure`
Modules used: `io_hdf5` `numerical_constants_prefixes`

type: `treemetadata`
Description: Structure that holds metadata for the trees.
Code lines: 8
Contained by: module `merger_tree_data_structure`

type: `unitsmetadata`
Description: A structure that holds metadata on units used.
Code lines: 5
Contained by: module `merger_tree_data_structure`

file: `objects.merger_trees.dump.F90`
Description: Contains a module which implements dumping of the structure of a merger tree to a file for plotting with DOT.
Code lines: 227

module: `merger_trees_dump`

Description:	Implements dumping of the structure of a merger tree to a file for plotting with DOT.	
Code lines:	205	
Contained by:	file `objects.merger_trees.dump.F90`	
Modules used:	`iso_c_binding`	`kind_numbers`
Used by:	subroutine `merger_tree_structure_dump`	subroutine `standardevolve`
	subroutine `dumptographvizoperate`	subroutine `regridtimesoperate`

subroutine: `merger_tree_dump`

Description:	Dumps the tree structure to a file in a format suitable for processing with DOT. Nodes are shown as circles if isolated or rectangles if satellites. Isolated nodes are connected to their descendent halo, while satellites are connected (by red lines) to their host halo. Optionally, a list of node indices to highlight can be specified.	
Code lines:	189	
Contained by:	module `merger_trees_dump`	
Modules used:	`galacticus_nodes`	`iso_varying_string`
	`merger_tree_walkers`	

file: `objects.nodes.F90`

Description:	Contains a module which implements an object hierarchy for nodes in merger trees and all of their constituent physical components.
Code lines:	101345

module: `galacticus_nodes`

Description: Implements an object hierarchy for nodes in merger trees and all of their constituent physical components. Defines an object class for merger trees. Defines an object class for the simulation universe. Contains custom functions for the standard basic component. Contains custom functions for the standard basic component. Contains custom functions for the simple black hole component. Contains custom functions for the standard black hole component. Contains custom functions for the standard disk component. Contains custom functions for the very simple disk component. Contains custom functions for the very simple size disk component. Contains custom functions for the bars dynamics statistics component. Contains custom functions for the Cole2000 formation times component. Contains custom functions for the cold mode hot halo component. Contains custom functions for the outflow tracking hot halo component. Contains custom functions for the standard hot halo component. Contains custom functions for the preset position component. Contains custom functions for the preset+orphans position component. Contains custom functions for the orbiting satellite component. Contains custom functions for the prest satellite component. Contains custom functions for the standard satellite component. Contains custom functions for the very simple satellite component. Contains custom functions for the standard spheroid component. Contains custom functions for the very simple spheroid component. Defines functions bound to the merger tree object class. Defines functions bound to the universe class.

Code lines: 101322

Contained by: file objects.nodes.F90

Modules used:

abundances_structure	chemical_abundances_structure
galacticus_error	hashes
histories	io_hdf5
iso_c_binding	iso_varying_string
kepler_orbits	kind_numbers
memory_management	numerical_constants_astronomical
pseudo_random	stellar_luminosities_structure
tensors	

Used by:

module accretion_halos	function naozbarkana2007accretedmass
function naozbarkana2007accretionrate	function naozbarkana2007branchhasbaryons
function naozbarkana2007failedaccretedmass	function naozbarkana2007failedaccretionrate
function naozbarkana2007filteredfraction	function naozbarkana2007filteredfractionrate
function coldmodechemicalmasses	function coldmodecoldmodefraction
function simpleaccretedmass	function simpleaccretionrate
function simplechemicalmasses	function simpleconstructorinternal
function simplefailedaccretedmass	function simplefailedaccretionrate

function simplefailedfraction
function bertschingeraccretionrate
function simpleaccretedmass
module accretion_disks
function fileconstructorinternal
module black_hole_binary_initial_separation
function spheroidradiusfractionseparationinitial
module black_hole_binary_recoil_velocities
function standardgrowthrate
function black_hole_frame_dragging_frequency_node
function black_hole_horizon_radius_node
function black_hole_isco_specific_angular_momentum
function black_hole_metric_a_factor_node
function black_hole_rotational_energy_spin_down_node
module chemical_reaction_rates
function dynamicaltimeconstructorinternal
module cooling_radii
function betaprofileradius
function isothermalconstructorinternal
function simpleconstructorinternal
function simpleradiusgrowthrate
module cooling_rates
function whitefrenk1991rate
function simplerate
function simplescalingrate
function velocitymaximumscalingrate
module cooling_freefall_times_available
function haloformationtimeavailable
module cooling_specific_angular_momenta
function meanangularmomentumspecific
function whitefrenk1991timeavailable
function bertschingeraccretedmass
module accretion_halo_totals
function simpleaccretionrate
module accretion_disk_spectra
function filespectrum
function volonteri2003separationinitial
function tidalradiusseparationinitial
module black_hole_binary_separations
function black_hole_eddington_accretion_rate
function black_hole_gravitational_radius
function black_hole_isco_radius_node
function black_hole_isco_specific_energy_node
function black_hole_metric_d_factor_node
function black_hole_static_radius_node
module cooling_cold_mode_infall_rates
function dynamicaltimeinfallrate
function betaprofileconstructorinternal
function betaprofileradiusgrowthrate
function isothermalradius
function simpleradius
function cole2000rate
function whitefrenk1991constructorinternal
function cutoffrate
function simplescalingconstructorinternal
function velocitymaximumscalingconstructorinternal
module freefall_radii
function haloformationconstructorparameters
module cooling_infall_radii
function constantrotationangularmomentumspecific
module cooling_times_available
function formationtimeconstructorparameters

function formationtimetimeavailable
function correa2015time
function wechsler2002time
module dark_matter_halos_mass_loss_rates
module dark_matter_halo_scales
function virialdensitycontrastdefinitionmeandensity
function virialdensitycontrastdefinitionvirialradius
function virialdensitycontrastdefinitionvirialvelocity
function virialdensitycontrastdefinitionvirialvelocitygrowthrate
function dark_matter_halo_angular_momentum
function dark_matter_halo_spin
module halo_spin_distributions
function nbodyerrorsdistributionaveraged
subroutine nbodyerrorstabulate
module dark_matter_profiles
function adiabaticgnedin2004massenclosed
function adiabaticgnedin2004velocitycircularsquaredgradient
function tidalspecificenergy
function tidalspecificenergyiseverywherezero
function twobodyrelaxationspecificenergygradient
function bullock2001concentration
function diemerkravtsov2014concentrationmean
module dark_matter_profiles_concentration
function klypin2015concentration
function munozcuartas2011concentration
function prada2011concentration
function wdmconcentration
function dark_matter_profile_mass_definition
function dark_matter_halo_formation_time
module dark_matter_halo_mass_accretion_histories
function zhao2009time
function vandenboschrate
function virialdensitycontrastdefinitionmeandensitygrowthrate
function virialdensitycontrastdefinitionvirialradiusgradient
function virialdensitycontrastdefinitionvirialvelocitygradient
subroutine assertpropertiesgettable
function dark_matter_halo_angular_momentum_growth_rate
function bett2007distribution
function nbodyerrorsdistribution
function nbodyerrorsdistributionfixedpoint
function lognormaldistribution
subroutine adiabaticgnedin2004computefactors
function adiabaticgnedin2004velocitycircularsquared
module dark_matter_profiles_generic
function tidalspecificenergygradient
function twobodyrelaxationspecificenergy
function twobodyrelaxationspecificenergyiseverywherezero
function correa2015concentration
function duttonmaccio2014concentration
function gao2008concentration
function ludlow2016fitconcentration
function nfw1996concentration
function schneider2015concentration
function zhao2009concentration
module dark_matter_profile_scales

function truncatedexponentialradiusenclosingmass
function node_branch_jump
subroutine event_halo_formation
module node_events_inter_tree
function node_pull_from_tree
function node_push_from_tree
subroutine node_promotion_index_shift
subroutine satellite_merging_remnant_compute
function node_subhalo_promotion
subroutine galacticus_calculations_reset_null
module galactic_filters
function ismmasspasses
function basicmasspasses
function formationtimepasses
function halomasspasses
function hostmassrangepasses
function nodemajormergerrecentpasses
function spheroidstellarmasspasses
function starformationratepasses
function stellarabsolutemagnitudespasses
function stellarapparentmagnitudespasses
function stellarmasspasses
function stellarmassmorphologypasses
function surveygeometrypasses
function efstathiou1982estimator
subroutine efstathiou1982timescale
function efstathiou1982tidalestimator
function efstathiou1982tidaltidaltensorradial
subroutine efstathiou1982tidaltimescale
module galactic_dynamics_bar_instabilities
function component_density
function galactic_structure_density
module galactic_structure_enclosed_masses
function component_enclosed_mass
function galactic_structure_enclosed_mass
function galactic_structure_radius_enclosing_density
function component_potential
function galactic_structure_potential
subroutine galactic_structure_potential_standard_reset
subroutine galactic_structure_radii_definition_decode
module galactic_structure_solvers
subroutine equilibriumsolveprederiativehook
function fixedconstructorinternal
subroutine radiussolve
function component_rotation_curve
function galactic_structure_rotation_curve
function component_rotation_curve_gradient
function galactic_structure_rotation_curve_gradient
function component_surface_density
function galactic_structure_surface_density
module galactic_structure_velocity_dispersions
subroutine galacticus_calculations_reset
subroutine meta_tree_timing_post_evolve
subroutine meta_tree_timing_pre_evolve
module output_analyses
function hivshalomassrelationpadmanabhan2017constructorinternal

subroutine correlationfunctionaccumulatehalo
module output_analysis_distribution_operators
function obreschkow2009ratio

subroutine volumefunction1danalyze

subroutine galacticus_extra_output_halo_fourier_profile
function squareisinlightcone
function squarereplicationcount
subroutine halo_model_projected_correlation
function enzohydrostaticdensitynormalization
function enzohydrostaticradialmoment

module hot_halo_mass_distributions

function patejloeb2015constructorinternal
function patejloeb2015densitylogslope
function patejloeb2015radialmoment

function ricotti2000constructorinternal
function betaprofileconstructorparameters
subroutine betaprofileinitialize
function betaprofilerotationnormalization
function growingconstructorinternal
module hot_halo_outflows_reincorporations
function halodynamicaltimerate

function velocitymaximumscalingrate
function font2008force
module hot_halo_ram_pressure_stripping_timescales
module hot_halo_temperature_profiles
module merger_tree_construction
module merger_trees_build_mass_resolution
module output_analysis_distribution_normalizers
module output_analysis_molecular_ratios
module output_analysis_property_operators
module output_analysis_weight_operators
module geometry_lightcones

function squareposition
function squarevelocity
module hot_halo_cold_mode_density_core_radii
function enzohydrostaticenclosedmass

function enzohydrostaticrotationnormalization
function hothalomassdistributionenclosedmass
function patejloeb2015density

function patejloeb2015enclosedmass
function patejloeb2015rotationnormalization
subroutine ricotti2000initialize
function betaprofileenclosedmass

function betaprofileradialmoment
module hot_halo_mass_distributions_core_radii
function growingradius
function henriques2013rate

function velocitymaximumscalingconstructorinternal
module hot_halo_ram_pressure_forces
module hot_halo_ram_pressure_stripping
function rampressureaccelerationtimescale
module merger_tree_branching
function buildconstruct
function scaledresolution

function halomassfunctionsample
subroutine cole2000build

function cole2000shouldfollowbranch
function fullyspecifiedconstruct
file merger_trees.construct.read.F90
subroutine readassignnamedproperties
subroutine readassignspinparameters
subroutine readassignuniqueidstoclones

subroutine readbuildisolatedparentpointers
function readconstruct
subroutine readcreatebranchjumpevent
subroutine readintertreemergetimeset
subroutine readscanformergers
subroutine readtimeuntilmergingsubresolution
function smoothaccretionconstruct
function nodepointer
function staterestoredconstruct
subroutine merger_tree_structure_dump
function historyconstructorinternal
function historytimeevolveto
function recordevolutiontimeevolveto
function satellitetimeevolveto
module merger_trees_evolve
subroutine standarddeadlockoutputtree
subroutine standardnodeeventsperform
subroutine standardtreeeventsperform
module merger_trees_evolve_node
subroutine standardintegrands
module merger_trees_merge_node
module merger_tree_operators
function augmentaccepttree
function augmentchildcount
subroutine augmentextendnonoverlapnodes
subroutine augmentnonoverlaplistadd
subroutine augmentoperate
subroutine augmentsortchildren
function conditionalmfbinweights
function conditionalmfconstructorinternal
subroutine deforestoperate
subroutine fixedmassconstruct
function cole2000criticaloverdensityupdate
module merger_trees_builders
function nodelookup
subroutine readassignmergers
subroutine readassignscaleradii
subroutine readassignsplitforestevents
subroutine readbuildchildandsiblinglinks
subroutine readbuildsubhalomasshistories
function readconstructorinternal
subroutine readcreatenodearray
subroutine readscanforbranchjumps
subroutine readscanforsubhalopromotions
module merger_tree_read_importers

subroutine mergertreestatefromfile
subroutine mergertreestatestore
subroutine merger_tree_dump_evolution
module merger_tree_timesteps
subroutine historystore
subroutine recordevolutionstore
function satelliteconstructorinternal
function simpletimeevolveto
file merger_trees.evolver.standard.F90
subroutine standardevolve
function standardtimeevolveto
subroutine merger_tree_initialize
subroutine standardevolve
function standardodes
subroutine singlelevelhierarchyprocess
subroutine assignorbitsoperate
function augmentbuildtreefromnode
subroutine augmentextendbyoverlap
function augmentnodecomparison
subroutine augmentnonoverlapreinsert
subroutine augmentsimpleinsert
function augmenttreestatistics
function conditionalmfbinweights2d
subroutine conditionalmfoperate

subroutine dumptographvizoperate

subroutine exportoperate

subroutine massaccretionhistoryoperate
subroutine outputrootmassesoperate

subroutine particulateoperate
subroutine profileroperate
subroutine prunebymassoperate
subroutine pruneclonesoperate
subroutine prunelightconevalidate
subroutine regridtimesoperate
subroutine merger_tree_structure_output
subroutine analyzeroutput
subroutine standardoutput
subroutine standardpropertynamesestablish
subroutine merger_tree_prune_uniqueify_ids
subroutine merger_trees_render_dump
file merger_trees.walkers.all_nodes.F90
file merger_trees.walkers.tree_construction.F90
function spindistributionevaluate
function icmszextract
function rateinfallcoldmodeextract
function ratecoolingextract
file nodes.property_extractor.density_contrasts.F90
function masshostextract
function luminositystellarextract
function massblackholeextract
function metallicityismextract
function projecteddensityextract
function rotationcurveextract
function satellitestatusconstructorinternal
function spinbullockconstructorinternal
function spinextract
function velocitymaximumextract
subroutine node_component_age_statistics_standard_inactive
subroutine node_component_age_statistics_standard_rate_compute
subroutine node_component_age_statistics_standard_scale_set

function massaccretionhistoryconstructorinternal
subroutine monotonizemassgrowthoperate
file merger_trees.operators.particulate.F90
subroutine perturbmassesoperate
subroutine prunebaryonsoperate
subroutine prunebytimeoperate
subroutine prunelightconeoperate
subroutine prunenonessentialoperate
subroutine selectwithinrangeoperate
module merger_tree_outputters
subroutine standardmakegroup
subroutine standardpropertiescount
subroutine merger_tree_prune_clean_branch
subroutine merger_tree_prune_unlink_parent
module merger_tree_walkers
function isolatednodesbranchnext
function sedfitevaluate

module node_property_extractors
function icmxrayluminosityextract
function concentrationextract
function densityprofileextract
function descendentsextract

function lmnstyemssnlineextract
function lmnstystllrchrltfll2000extract
function masshaloextract
function mostmassiveprogenitorextract
function redshiftlastisolatedextract
function satelliteorbitalextremaextract
function satellitestatusextract

function spinbullockextract
function velocitydispersionextract
subroutine merger_tree_dump
subroutine node_component_age_statistics_standard_initialize
subroutine node_component_age_statistics_standard_satellite_merging
subroutine node_component_basic_extended_bertschinger_solver

subroutine node_component_basic_-extended_bindings
function node_component_basic_-extended_mass_bertschinger
function node_component_basic_-extended_radius_turnaround
subroutine node_component_basic_-extended_thread_initialize
subroutine node_component_basic_-extended_thread_uninitialize
subroutine node_component_basic_-standard_extended_initialize
subroutine node_component_basic_-standard_extended_node_merger
subroutine node_component_basic_-standard_extended_promote
subroutine node_component_basic_-standard_extended_rate_compute
subroutine node_component_basic_-standard_extended_scale_set
function node_component_basic_-standard_extended_unresolved_mass
subroutine node_component_basic_-extended_tracking_promote
subroutine node_component_basic_-extended_tree_tracking_initialize
subroutine node_component_basic_non_-evolving_promote
subroutine node_component_basic_non_-evolving_rate_compute
subroutine node_component_basic_non_-evolving_scale_set
subroutine node_component_basic_-standard_plausibility
subroutine node_component_basic_-standard_post_step
subroutine node_component_basic_-standard_promote
subroutine node_component_basic_-standard_rate_compute
subroutine node_component_basic_-standard_scale_set
subroutine node_component_basic_-standard_stop_accretion
subroutine node_component_basic_-standard_tree_initialize
function node_component_basic_-standard_unresolved_mass
subroutine node_component_basic_-standard_tracking_promote
subroutine node_component_basic_-standard_tree_tracking_initialize
subroutine node_component_black_hole_-noncentral_merge_black_holes
subroutine node_component_black_hole_-noncentral_rate_compute
function node_component_black_hole_-noncentral_recoil_escapes
subroutine node_component_black_hole_-noncentral_scale_set
subroutine node_component_black_hole_-noncentral_thread_initialize
subroutine node_component_black_hole_-noncentral_thread_uninitialize
subroutine node_component_black_hole_-noncentral_triple_interaction
subroutine node_component_black_hole_-simple_create
subroutine node_component_black_hole_-simple_initialize
function node_component_black_hole_-simple_matches
subroutine node_component_black_hole_-simple_output
subroutine node_component_black_hole_-simple_output_count
subroutine node_component_black_hole_-simple_output_names
subroutine node_component_black_hole_-simple_rate_compute
subroutine node_component_black_hole_-simple_satellite_merging
subroutine node_component_black_hole_-simple_scale_set
subroutine node_component_black_hole_-simple_thread_initialize
subroutine node_component_black_hole_-simple_thread_uninitialize

function node_component_black_hole_simple_potential
function node_component_black_hole_simple_rotation_curve
function node_component_black_hole_simple_rotation_curve_gradient
function hot_mode_fraction
function node_component_black_hole_standard_accretion_rate
subroutine node_component_black_hole_standard_create
subroutine node_component_black_hole_standard_initialize
subroutine node_component_black_hole_standard_mass_accretion_rate
function node_component_black_hole_standard_matches
subroutine node_component_black_hole_standard_output
subroutine node_component_black_hole_standard_output_count
subroutine node_component_black_hole_standard_output_merger
subroutine node_component_black_hole_standard_output_names
subroutine node_component_black_hole_standard_output_properties
subroutine node_component_black_hole_standard_post_evolve
function node_component_black_hole_standard_radiative_efficiency
subroutine node_component_black_hole_standard_rate_compute
function node_component_black_hole_standard_recoil_escapes
subroutine node_component_black_hole_standard_satellite_merging
subroutine node_component_black_hole_standard_scale_set
subroutine node_component_black_hole_standard_thread_initialize
subroutine node_component_black_hole_standard_thread_uninitialize
function node_component_black_hole_standard_potential
function node_component_black_hole_standard_rotation_curve
function node_component_black_hole_standard_rotation_curve_gradient
module node_component_dark_matter_profile_scale
subroutine node_component_dark_matter_profile_scale_plausibility
subroutine node_component_dark_matter_profile_scale_promote
subroutine node_component_dark_matter_profile_scale_rate_compute
function node_component_dark_matter_profile_scale_scale
subroutine node_component_dark_matter_profile_scale_scale_set
subroutine node_component_dark_matter_profile_scale_thread_initialize
subroutine node_component_dark_matter_profile_scale_thread_uninitialize
subroutine node_component_dark_matter_profile_scale_tree_initialize
subroutine node_component_dark_matter_profile_scale_tree_output
subroutine node_component_dark_matter_profile_scale_preset_promote
subroutine node_component_dark_matter_profile_scale_preset_rate_compute
subroutine node_component_dark_matter_profile_scale_preset_scale_set
subroutine node_component_dark_matter_profile_scale_preset_tree_initialize
module node_component_dark_matter_profile_scale_shape
subroutine node_component_dark_matter_profile_scale_shape_promote
subroutine node_component_dark_matter_profile_scale_shape_rate_compute
subroutine node_component_dark_matter_profile_scale_shape_scale_set
subroutine node_component_dark_matter_profile_scale_shape_thread_init

subroutine node_component_dark_matter_profile_scale_shape_thread_uninit
subroutine node_component_dark_matter_profile_scale_shape_tree_initialize
subroutine node_component_dark_matter_profile_scale_shape_tree_output
subroutine node_component_disk_standard_calculation_reset
subroutine node_component_disk_standard_create
subroutine node_component_disk_standard_final_state
subroutine node_component_disk_standard_inactive
subroutine node_component_disk_standard_initialize
subroutine node_component_disk_standard_post_evolve
subroutine node_component_disk_standard_post_step
subroutine node_component_disk_standard_pre_evolve
function node_component_disk_standard_radius_solve
subroutine node_component_disk_standard_radius_solve_set
subroutine node_component_disk_standard_radius_solver
subroutine node_component_disk_standard_radius_solver_plausibility
subroutine node_component_disk_standard_rate_compute
subroutine node_component_disk_standard_satellite_merging
subroutine node_component_disk_standard_scale_set
subroutine node_component_disk_standard_star_formation_history_output
function node_component_disk_standard_star_formation_rate
subroutine node_component_disk_standard_thread_initialize
subroutine node_component_disk_standard_thread_uninitialize
function node_component_disk_standard_velocity
subroutine node_component_disk_standard_velocity_set
module node_component_disk_very_simple
subroutine node_component_disk_very_simple_analytic_solver
subroutine node_component_disk_very_simple_create
subroutine node_component_disk_very_simple_initialize
subroutine node_component_disk_very_simple_post_evolve
subroutine node_component_disk_very_simple_post_step
subroutine node_component_disk_very_simple_pre_evolve
subroutine node_component_disk_very_simple_rate_compute
subroutine node_component_disk_very_simple_rates
subroutine node_component_disk_very_simple_satellite_merging
subroutine node_component_disk_very_simple_scale_set
function node_component_disk_very_simple_sfr
subroutine node_component_disk_very_simple_thread_initialize
subroutine node_component_disk_very_simple_thread_uninitialize
subroutine node_component_disk_very_simple_size_initialize
function node_component_disk_very_simple_size_radius
subroutine node_component_disk_very_simple_size_radius_set
subroutine node_component_disk_very_simple_size_radius_solver
subroutine node_component_disk_very_simple_size_radius_solver_plausibility
subroutine node_component_disk_very_simple_size_thread_initialize

subroutine node_component_disk_very_simple_size_thread_uninitialize
subroutine node_component_disk_very_simple_size_velocity_set
subroutine node_component_dynamics_statistics_bars_output
subroutine node_component_dynamics_statistics_bars_record
subroutine node_component_dynamics_statistics_bars_thread_uninitialize
subroutine node_component_formation_time_cole2000_node_promotion
subroutine node_component_formation_time_cole2000_tree_initialize
subroutine node_component_host_history_standard_merger_tree_init
subroutine node_component_hot_halo_cold_mode_formation
subroutine node_component_hot_halo_cold_mode_node_merger
subroutine node_component_hot_halo_cold_mode_promote
subroutine node_component_hot_halo_cold_mode_rate_compute
subroutine node_component_hot_halo_cold_mode_scale_set
subroutine node_component_hot_halo_cold_mode_thread_uninitialize
function node_component_hot_halo_cold_mode_density_task
function node_component_hot_halo_cold_mode_rotation_curve_gradient_task
subroutine node_component_hot_halo_outflow_tracking_rate_compute
subroutine node_component_hot_halo_outflow_tracking_thread_initialize
subroutine node_component_hot_halo_standard_cooling_rate
subroutine node_component_hot_halo_standard_formation
subroutine node_component_hot_halo_standard_hot_gas_all_rate
subroutine node_component_hot_halo_standard_initializor
function node_component_disk_very_simple_size_velocity
subroutine node_component_dynamics_statistics_bars_initialize
subroutine node_component_dynamics_statistics_bars_rate_compute
subroutine node_component_dynamics_statistics_bars_thread_initialize
subroutine node_component_formation_time_cole2000_create
subroutine node_component_formation_time_cole2000_rate_compute
module node_component_formation_times_mass_fraction
subroutine node_component_host_history_standard_update_history
subroutine node_component_hot_halo_cold_mode_initialize
subroutine node_component_hot_halo_cold_mode_outflow_return
subroutine node_component_hot_halo_cold_mode_push_to_cooling_pipes
subroutine node_component_hot_halo_cold_mode_satellite_merging
subroutine node_component_hot_halo_cold_mode_thread_initialize
subroutine node_component_hot_halo_cold_mode_tree_initialize
function node_component_hot_halo_cold_mode_enclosed_mass_task
function node_component_hot_halo_cold_mode_rotation_curve_task
subroutine node_component_hot_halo_outflow_tracking_scale_set
subroutine node_component_hot_halo_outflow_tracking_thread_uninitialize
subroutine node_component_hot_halo_standard_create
subroutine node_component_hot_halo_standard_heat_source
subroutine node_component_hot_halo_standard_initialize
subroutine node_component_hot_halo_standard_mass_sink

subroutine node_component_hot_halo_standard_node_merger
function node_component_hot_halo_standard_outer_radius
function node_component_hot_halo_standard_outer_radius_growth_rate
subroutine node_component_hot_halo_standard_outflow_return
function node_component_hot_halo_standard_outflow_stripped_fraction
subroutine node_component_hot_halo_standard_outflowing_abundances_rate
subroutine node_component_hot_halo_standard_outflowing_ang_mom_rate
subroutine node_component_hot_halo_standard_outflowing_mass_rate
subroutine node_component_hot_halo_standard_post_evolve
subroutine node_component_hot_halo_standard_post_step
subroutine node_component_hot_halo_standard_pre_evolve
subroutine node_component_hot_halo_standard_promote
subroutine node_component_hot_halo_standard_push_from_halo
subroutine node_component_hot_halo_standard_push_to_cooling_pipes
subroutine node_component_hot_halo_standard_rate_compute
subroutine node_component_hot_halo_standard_reset
subroutine node_component_hot_halo_standard_satellite_merging
subroutine node_component_hot_halo_standard_scale_set
subroutine node_component_hot_halo_standard_strip_gas_rate
subroutine node_component_hot_halo_standard_thread_initialize
subroutine node_component_hot_halo_standard_thread_uninitialize
subroutine node_component_hot_halo_standard_tree_initialize
subroutine node_component_hot_halo_very_simple_cooling_rate
subroutine node_component_hot_halo_very_simple_create
subroutine node_component_hot_halo_very_simple_initialize
subroutine node_component_hot_halo_very_simple_node_merger
function node_component_hot_halo_very_simple_outer_radius
subroutine node_component_hot_halo_very_simple_outflowing_abundances_rate
subroutine node_component_hot_halo_very_simple_outflowing_mass_rate
subroutine node_component_hot_halo_very_simple_post_evolve
subroutine node_component_hot_halo_very_simple_promote
subroutine node_component_hot_halo_very_simple_push_to_cooling_pipes
subroutine node_component_hot_halo_very_simple_rate_compute
subroutine node_component_hot_halo_very_simple_reset
subroutine node_component_hot_halo_very_simple_satellite_merging
subroutine node_component_hot_halo_very_simple_scale_set
subroutine node_component_hot_halo_very_simple_thread_initialize
subroutine node_component_hot_halo_very_simple_thread_uninitialize
subroutine node_component_hot_halo_very_simple_tree_initialize
subroutine node_component_hot_halo_vs_delayed_initialize
subroutine node_component_hot_halo_vs_delayed_node_merger
subroutine node_component_hot_halo_vs_delayed_outflowing_abundances_rate
subroutine node_component_hot_halo_vs_delayed_outflowing_mass_rate
subroutine node_component_hot_halo_vs_delayed_post_evolve

subroutine node_component_hot_halo_vs_delayed_promote
subroutine node_component_hot_halo_vs_delayed_rate_compute
subroutine node_component_hot_halo_vs_delayed_satellite_merging
subroutine node_component_hot_halo_vs_delayed_scale_set
subroutine node_component_hot_halo_vs_delayed_tree_initialize
subroutine node_component_indices_standard_merger_tree_init
subroutine node_component_inter_output_standard_rate_compute
subroutine node_component_inter_output_standard_reset
subroutine node_component_inter_output_standard_satellite_merging
subroutine node_component_inter_output_standard_scale_set
subroutine node_component_interoutput_standard_thread_initialize
subroutine node_component_interoutput_standard_thread_uninitialize
subroutine node_component_mass_flow_statistics_standard_extra_output
subroutine node_component_mass_flow_statistics_standard_merger_tree_init
subroutine node_component_mass_flow_statistics_standard_rate_compute
subroutine node_component_mass_flow_statistics_standard_scale_set
subroutine node_component_mass_flow_statistics_standard_thread_initialize
subroutine node_component_mass_flow_statistics_standard_thread_uninit
subroutine node_component_merging_statistics_major_node_promotion
subroutine node_component_merging_statistics_major_output
subroutine node_component_merging_statistics_major_satellite_merging
function node_component_merging_statistics_recent_matches
subroutine node_component_merging_statistics_recent_merger_tree_init
subroutine node_component_merging_statistics_recent_node_merger
subroutine node_component_merging_statistics_recent_node_promotion
subroutine node_component_merging_statistics_recent_output
subroutine node_component_merging_statistics_recent_output_count
subroutine node_component_merging_statistics_recent_output_names
subroutine node_component_merging_statistics_recent_thread_initialize
subroutine node_component_merging_statistics_recent_thread_uninitialize
module node_component_merging_statistics_standard
function node_component_merging_statistics_standard_hierarchy_level
function node_component_merging_statistics_standard_hlm
subroutine node_component_merging_statistics_standard_merger_tree_init
subroutine node_component_merging_statistics_standard_node_merger
subroutine node_component_merging_statistics_standard_node_promotion
subroutine node_component_merging_statistics_standard_reset_hierarchy
subroutine node_component_merging_statistics_standard_satellite_merging
subroutine node_component_merging_statistics_standard_thread_initialize
subroutine node_component_merging_statistics_standard_thread_uninitialize

function node_component_nbody_generic_add_integer_property
function node_component_nbody_generic_add_real_property
subroutine node_component_nbody_generic_initialize
subroutine node_component_nbody_generic_output

subroutine node_component_nbody_generic_output_count
subroutine node_component_nbody_generic_output_names
subroutine node_component_nbody_generic_promote
subroutine node_component_nbody_generic_set_integer_property
subroutine node_component_nbody_generic_set_real_property
subroutine node_component_position_preset_initialize
subroutine node_component_position_preset_inter_tree_insert
subroutine node_component_position_preset_move
subroutine node_component_position_preset_node_promotion
subroutine node_component_position_preset_orphans_initialize
function node_component_position_preset_orphans_position_orphan
subroutine node_component_position_preset_orphans_thread_initialize
subroutine node_component_position_preset_orphans_thread_uninitialize
function node_component_position_preset_orphans_velocity_orphan
subroutine node_component_position_trace_dark_matter_assign
subroutine node_component_position_trace_dark_matter_initialize
subroutine node_component_position_trace_dark_matter_thread_initialize
subroutine node_component_position_trace_dark_matter_thread_uninitialize
subroutine node_component_position_trace_dark_matter_update
subroutine node_component_satellite_orbiting_bound_mass_initialize
subroutine node_component_satellite_orbiting_create
subroutine node_component_satellite_orbiting_initialize
subroutine node_component_satellite_orbiting_rate_compute
subroutine node_component_satellite_orbiting_scale_set
subroutine node_component_satellite_orbiting_thread_initialize
subroutine node_component_satellite_orbiting_thread_uninitialize
subroutine node_component_satellite_orbiting_tree_initialize
subroutine node_component_satellite_orbiting_trigger_merger
subroutine node_component_satellite_orbiting_virial_orbit_set
subroutine node_component_satellite_preset_inter_tree_attach
subroutine node_component_satellite_preset_inter_tree_insert
subroutine node_component_satellite_preset_inter_tree_postprocess
subroutine node_component_satellite_preset_orphanize
subroutine node_component_satellite_preset_promote
subroutine node_component_satellite_preset_rate_compute
subroutine node_component_satellite_preset_satellite_host_change
subroutine node_component_satellite_standard_create
subroutine node_component_satellite_standard_halo_formation_task
subroutine node_component_satellite_standard_inactive
subroutine node_component_satellite_standard_initialize
subroutine node_component_satellite_standard_rate_compute
subroutine node_component_satellite_standard_scale_set
subroutine node_component_satellite_standard_thread_initialize
subroutine node_component_satellite_standard_thread_uninitialize

subroutine node_component_satellite_standard_tree_initialize
subroutine node_component_satellite_standard_virial_orbit_set
subroutine node_component_satellite_very_simple_halo_formation_task
subroutine node_component_satellite_very_simple_thread_initialize
subroutine node_component_satellite_very_simple_tree_initialize
subroutine node_component_spheroid_standard_inactive
subroutine node_component_spheroid_standard_initializor
subroutine node_component_spheroid_standard_post_evolve
subroutine node_component_spheroid_standard_pre_evolve
subroutine node_component_spheroid_standard_radius_solve_set
subroutine node_component_spheroid_standard_radius_solver_plausibility
subroutine node_component_spheroid_standard_satellite_merging
subroutine node_component_spheroid_standard_star_formation_history_extend
subroutine node_component_spheroid_standard_star_formation_history_rate
subroutine node_component_spheroid_standard_stellar_prprts_history_extend
subroutine node_component_spheroid_standard_thread_initialize
function node_component_spheroid_standard_velocity_solve
subroutine node_component_spheroid_very_simple_create
subroutine node_component_spheroid_very_simple_post_evolve
subroutine node_component_spheroid_very_simple_pre_evolve
subroutine node_component_spheroid_very_simple_radius_set
function node_component_satellite_standard_virial_orbit
subroutine node_component_satellite_very_simple_create
subroutine node_component_satellite_very_simple_rate_compute
subroutine node_component_satellite_very_simple_thread_uninitialize
subroutine node_component_spheroid_standard_energy_gas_input_rate
subroutine node_component_spheroid_standard_initialize
subroutine node_component_spheroid_standard_mass_gas_sink_rate
subroutine node_component_spheroid_standard_post_step
function node_component_spheroid_standard_radius_solve
subroutine node_component_spheroid_standard_radius_solver
subroutine node_component_spheroid_standard_rate_compute
subroutine node_component_spheroid_standard_scale_set
subroutine node_component_spheroid_standard_star_formation_history_output
function node_component_spheroid_standard_star_formation_rate
subroutine node_component_spheroid_standard_stellar_prprts_history_rate
subroutine node_component_spheroid_standard_thread_uninitialize
subroutine node_component_spheroid_standard_velocity_solve_set
subroutine node_component_spheroid_very_simple_initialize
subroutine node_component_spheroid_very_simple_post_step
function node_component_spheroid_very_simple_radius
subroutine node_component_spheroid_very_simple_radius_solver

subroutine node_component_spheroid_very_simple_radius_solver_plausibility
subroutine node_component_spheroid_very_simple_rate_compute
subroutine node_component_spheroid_very_simple_rates
subroutine node_component_spheroid_very_simple_satellite_merging
subroutine node_component_spheroid_very_simple_scale_set
function node_component_spheroid_very_simple_sfr
subroutine node_component_spheroid_very_simple_thread_initialize
subroutine node_component_spheroid_very_simple_thread_uninitialize
function node_component_spheroid_very_simple_velocity
subroutine node_component_spheroid_very_simple_velocity_set
subroutine node_component_spin_vitvitska_bindings
subroutine node_component_spin_vitvitska_branch_initialize
subroutine node_component_spin_vitvitska_initialize_spins
subroutine node_component_spin_vitvitska_promote
subroutine node_component_spin_vitvitska_rate_compute
subroutine node_component_spin_vitvitska_scale_set
function node_component_spin_vitvitska_spin
function node_component_spin_vitvitska_spin_growth_rate
function node_component_spin_vitvitska_spin_vector
subroutine node_component_spin_vitvitska_thread_initialize
subroutine node_component_spin_vitvitska_thread_uninitialize
function orbital_angular_momentum
subroutine node_component_spin_preset_initialize
subroutine node_component_spin_preset_promote
subroutine node_component_spin_preset_rate_compute
subroutine node_component_spin_preset_scale_set
subroutine node_component_spin_preset3d_initialize
subroutine node_component_spin_preset3d_promote
subroutine node_component_spin_preset3d_rate_compute
subroutine node_component_spin_preset3d_scale_set
subroutine node_component_spin_random_initialize_spins
subroutine node_component_spin_random_promote
subroutine node_component_spin_random_thread_initialize
subroutine node_component_spin_random_thread_uninitialize
module radiation_fields
subroutine intergalacticbackgroundinternaluniversepreevolve
function intergalacticbackgroundinternalupdate
module ram_pressure_stripping_mass_loss_rate_disks
function simpleratemassloss
module ram_pressure_stripping_mass_loss_rate_spheroids
function simpleratemassloss
function chandrasekhar1943acceleration
module satellite_dynamical_friction
module satellite_merging_mass_movements

function superwindoutflowrate

function superwindoutflowrate
subroutine insitucreate
subroutine insiturate
subroutine metallicitysplitcreate
subroutine metallicitysplitrate
module star_formation_rate_surface_-density_disks
subroutine krumholz2009computefactors
function extendedschmidtrate
module star_formation_timescales_disks

function dynamicaltimetimescale
function intgrtdsurfacedensitytimescale
module star_formation_timescales_-spheroids
function dynamicaltimetimescale
module statistics_nbody_halo_mass_-errors
function sohalofindererrorfractional
function nullcorrelation
function powerlawerrorfractional
subroutine instantaneousrates

subroutine noninstantaneousrates
function diskspheroidselect
function environmentalgradientmass
function environmentalvalue
module excursion_sets_first_crossings
function lognormaloverdensitylinear
function normaloverdensitynonlinear
function environmentaverageddifferential
function fofbiasdifferential

function virial_density_contrast_-percolation_solver
file tasks.evolve_forests.F90

subroutine evolveforestsperform

subroutine excursionsetsperform

module star_formation_feedback_-expulsion_spheroids
module star_formation_histories
subroutine insituoutput
subroutine insitusatellitemerger
subroutine metallicitysplitoutput
function blitz2006rate
function kennicuttschmidtrate

function krumholz2009unchanged
function baugh2005timescale
function dynamicaltimeconstructorinternal
function haloscalingtimescale
function velocitymaxscalingtimescale
function dynamicaltimeconstructorinternal
function velocitymaxscalingtimescale
function sohalofindercorrelation

function trenti2010correlation
function powerlawcorrelation
module stellar_population_properties
subroutine noninstantaneoushistorycreate
module stellar_population_selectors
module cosmological_density_field
function environmentalgradienttime
module excursion_sets_barriers
module dark_matter_halo_biases
function normaloverdensitylinear
module halo_mass_functions
function errorconvolveddifferential

module virial_density_contrast_-percolation_utilities
function catalogprojectedcorrelationfunctionconstructorparameters
function evolveforestsconstructorparameters
function excursionsetsconstructorparameters
function halomassfunctionconstructorparameters

subroutine halomassfunctionperform	function halomodelprojectedcorrelationfunctionconstructorparameters
function halomodelgenerateconstructorparameters	subroutine halomodelgenerateperform
function halospindistributionconstructorparameters	subroutine halospindistributionperform
function posteriorsampleconstructorparameters	program test_diemerkravtsov2014_concentration
program test_nfw96_concentration_dark_energy	program test_prada2011_concentration
program test_zhao2009_flat	program test_zhao2009_dark_energy
program test_zhao2009_open	program test_accretion_disks
program tests_bug745815	program test_correa2015_concentration
program test_dark_matter_halo_radius_enclosing_mass	program test_dark_matter_profiles
program test_dark_matter_profiles_generic	program test_dark_matter_profiles_heated
program test_correa2015_mah	program test_nodes
subroutine test_node_task	function testfuncdouble0
subroutine testvoidfunc	program tests_tree_branch_destroy
module tidal_stripping_mass_loss_rate_disks	function simpleratemassloss
module tidal_stripping_mass_loss_rate_spheroids	function simpleratemassloss
module universe_operators	file universe.operators.intergalactic_medium_state_evolve.F90
subroutine intergalacticmediumstateevolveoperate	function intergalacticmediumstateevolveupdate

subroutine: agestatisticscreatebyinterrupt

Description: Create the ageStatistics component of self via an interrupt.
Code lines: 10
Contained by: module galacticus_nodes

function: agestatisticsdiskintegratedsfr

Description: Returns the default value for the diskIntegratedSFR property for the ageStatistics component class.
Code lines: 17
Contained by: module galacticus_nodes

function: agestatisticsdiskintegratedsfrattributematch

Description: Return a text list of component implementations in the ageStatistics class that have the desired attributes for the diskIntegratedSFR property
Code lines: 28
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: agestatisticsdiskintegratedsfrisgettable

Description: Returns true if the diskIntegratedSFR property is gettable for the ageStatistics component class.

Code lines: 9
Contained by: module galacticus_nodes

subroutine: agestatisticsdiskintegratedsfrrate
Description: Accept a rate set for the diskIntegratedSFR property of the ageStatistics component class. Trigger an interrupt to create the component.
Code lines: 21
Contained by: module galacticus_nodes
Modules used: galacticus_error

function: agestatisticsdiskintegratedsfrrateget
Description: Returns a zero rate for the diskIntegratedSFR property for the ageStatistics component class.
Code lines: 13
Contained by: module galacticus_nodes

function: agestatisticsdisktimeweightedintegratedsfr
Description: Returns the default value for the diskTimeWeightedIntegratedSFR property for the ageStatistics component class.
Code lines: 17
Contained by: module galacticus_nodes

function: agestatisticsdisktimeweightedintegratedsfrattributematch
Description: Return a text list of component implementations in the ageStatistics class that have the desired attributes for the diskTimeWeightedIntegratedSFR property
Code lines: 28
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: agestatisticsdisktimeweightedintegratedsfrisgettable
Description: Returns true if the diskTimeWeightedIntegratedSFR property is gettable for the ageStatistics component class.
Code lines: 9
Contained by: module galacticus_nodes

subroutine: agestatisticsdisktimeweightedintegratedsfrrate
Description: Accept a rate set for the diskTimeWeightedIntegratedSFR property of the ageStatistics component class. Trigger an interrupt to create the component.
Code lines: 21
Contained by: module galacticus_nodes
Modules used: galacticus_error

function: agestatisticsdisktimeweightedintegratedsfrrateget
Description: Returns a zero rate for the diskTimeWeightedIntegratedSFR property for the ageStatistics component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: agestatisticsnulloutputcount
Description: Increment the count of properties to output for a null implementation of the ageStatistics component.

Code lines: 22
Contained by: module galacticus_nodes

subroutine: agestatisticsnulloutputnames
Description: Return the names of properties to output for a null implementation of the ageStatistics component.
Code lines: 39
Contained by: module galacticus_nodes

subroutine: agestatisticsoutput
Description: Populate output buffers with properties to output for a ageStatistics component.
Code lines: 40
Contained by: module galacticus_nodes
Modules used: multi_counters

subroutine: agestatisticsoutputcount
Description: Increment the count of properties to output for a generic ageStatistics component.
Code lines: 15
Contained by: module galacticus_nodes

subroutine: agestatisticsoutputnames
Description: Establish the names of properties to output for a generic ageStatistics component.
Code lines: 20
Contained by: module galacticus_nodes

subroutine: agestatisticspostoutput
Description: Perform post-output processing of a ageStatistics component.
Code lines: 14
Contained by: module galacticus_nodes

function: agestatisticsspheroidintegratedsfr
Description: Returns the default value for the spheroidIntegratedSFR property for the ageStatistics component class.
Code lines: 17
Contained by: module galacticus_nodes

function: agestatisticsspheroidintegratedsfrattributematch
Description: Return a text list of component implementations in the ageStatistics class that have the desired attributes for the spheroidIntegratedSFR property
Code lines: 28
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: agestatisticsspheroidintegratedsfrisgettable
Description: Returns true if the spheroidIntegratedSFR property is gettable for the ageStatistics component class.
Code lines: 9
Contained by: module galacticus_nodes

subroutine: agestatisticsspheroidintegratedsfrrate

Description: Accept a rate set for the `spheroidIntegratedSFR` property of the `ageStatistics` component class. Trigger an interrupt to create the component.
Code lines: 21
Contained by: module galacticus_nodes
Modules used: galacticus_error

function: `agestatisticsspheroidintegratedsfrrateget`
Description: Returns a zero rate for the `spheroidIntegratedSFR` property for the `ageStatistics` component class.
Code lines: 13
Contained by: module galacticus_nodes

function: `agestatisticsspheroidtimeweightedintegratedsfr`
Description: Returns the default value for the `spheroidTimeWeightedIntegratedSFR` property for the `ageStatistics` component class.
Code lines: 17
Contained by: module galacticus_nodes

function: `agestatisticsspheroidtimeweightedintegratedsfrattributematch`
Description: Return a text list of component implementations in the `ageStatistics` class that have the desired attributes for the `spheroidTimeWeightedIntegratedSFR` property
Code lines: 28
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: `agestatisticsspheroidtimeweightedintegratedsfrisgettable`
Description: Returns true if the `spheroidTimeWeightedIntegratedSFR` property is gettable for the `ageStatistics` component class.
Code lines: 9
Contained by: module galacticus_nodes

subroutine: `agestatisticsspheroidtimeweightedintegratedsfrrate`
Description: Accept a rate set for the `spheroidTimeWeightedIntegratedSFR` property of the `ageStatistics` component class. Trigger an interrupt to create the component.
Code lines: 21
Contained by: module galacticus_nodes
Modules used: galacticus_error

function: `agestatisticsspheroidtimeweightedintegratedsfrrateget`
Description: Returns a zero rate for the `spheroidTimeWeightedIntegratedSFR` property for the `ageStatistics` component class.
Code lines: 13
Contained by: module galacticus_nodes

function: `agestatisticsstandarddiskintegratedsfrcount`
Description: Return a count of the number of scalar properties in the `diskIntegratedSFR` property of an `standard` implementation of the `ageStatistics` component class.
Code lines: 12
Contained by: module galacticus_nodes

function: `agestatisticsstandarddiskintegratedsfrget`
Description: Get the `diskIntegratedSFR` property of an `standard` implementation of the `ageStatistics` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `agestatisticsstandarddiskintegratedsfrjcbnzr`
Description: Indicate that the `diskIntegratedSFR` property of an `standard` implementation of the `ageStatistics` component class is inactive for differential equation solving.
Code lines: 12
Contained by: module `galacticus_nodes`

subroutine: `agestatisticsstandarddiskintegratedsfrrate`
Description: Accumulate to the rate of change of the `diskIntegratedSFR` property of an `standard` implementation of the `ageStatistics` component class.
Code lines: 31
Contained by: module `galacticus_nodes`

function: `agestatisticsstandarddiskintegratedsfrrateget`
Description: Get the rate of change of the `diskIntegratedSFR` property of an `standard` implementation of the `ageStatistics` component class.
Code lines: 23
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `agestatisticsstandarddiskintegratedsfrscale`
Description: Set the absolute scale of the `diskIntegratedSFR` property of an `standard` implementation of the `ageStatistics` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

subroutine: `agestatisticsstandarddiskintegratedsfrset`
Description: Set the `diskIntegratedSFR` property of an `standard` implementation of the `ageStatistics` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `agestatisticsstandarddisktimeweightedintegratedsfrcount`
Description: Return a count of the number of scalar properties in the `diskTimeWeightedIntegratedSFR` property of an `standard` implementation of the `ageStatistics` component class.
Code lines: 12
Contained by: module `galacticus_nodes`

function: `agestatisticsstandarddisktimeweightedintegratedsfrget`
Description: Get the `diskTimeWeightedIntegratedSFR` property of an `standard` implementation of the `ageStatistics` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: agestatisticsstandarddisktimeweightedintegratedsfrjcbnzr
Description: Indicate that the diskTimeWeightedIntegratedSFR property of an standard implementation of the ageStatistics component class is inactive for differential equation solving.
Code lines: 12
Contained by: module galacticus_nodes

subroutine: agestatisticsstandarddisktimeweightedintegratedsfrrate
Description: Accumulate to the rate of change of the diskTimeWeightedIntegratedSFR property of an standard implementation of the ageStatistics component class.
Code lines: 31
Contained by: module galacticus_nodes

function: agestatisticsstandarddisktimeweightedintegratedsfrrateget
Description: Get the rate of change of the diskTimeWeightedIntegratedSFR property of an standard implementation of the ageStatistics component class.
Code lines: 23
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: agestatisticsstandarddisktimeweightedintegratedsfrscale
Description: Set the absolute scale of the diskTimeWeightedIntegratedSFR property of an standard implementation of the ageStatistics component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: agestatisticsstandarddisktimeweightedintegratedsfrset
Description: Set the diskTimeWeightedIntegratedSFR property of an standard implementation of the ageStatistics component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: agestatisticsstandardoutputcount
Description: Increment the count of properties to output for a standard implementation of the ageStatistics component.
Code lines: 21
Contained by: module galacticus_nodes

subroutine: agestatisticsstandardoutputnames
Description: Return the names of properties to output for a standard implementation of the ageStatistics component.
Code lines: 55
Contained by: module galacticus_nodes

function: agestatisticsstandardspheroidintegratedsfrcount
Description: Return a count of the number of scalar properties in the spheroidIntegratedSFR property of an standard implementation of the ageStatistics component class.
Code lines: 12
Contained by: module galacticus_nodes

function: `agestatisticsstandardspheroidintegratedsfrget`
Description: Get the `spheroidIntegratedSFR` property of an `standard` implementation of the `ageStatistics` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `agestatisticsstandardspheroidintegratedsfrjcbnzr`
Description: Indicate that the `spheroidIntegratedSFR` property of an `standard` implementation of the `ageStatistics` component class is inactive for differential equation solving.
Code lines: 12
Contained by: module `galacticus_nodes`

subroutine: `agestatisticsstandardspheroidintegratedsfrrate`
Description: Accumulate to the rate of change of the `spheroidIntegratedSFR` property of an `standard` implementation of the `ageStatistics` component class.
Code lines: 31
Contained by: module `galacticus_nodes`

function: `agestatisticsstandardspheroidintegratedsfrrateget`
Description: Get the rate of change of the `spheroidIntegratedSFR` property of an `standard` implementation of the `ageStatistics` component class.
Code lines: 23
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `agestatisticsstandardspheroidintegratedsfrscale`
Description: Set the absolute scale of the `spheroidIntegratedSFR` property of an `standard` implementation of the `ageStatistics` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

subroutine: `agestatisticsstandardspheroidintegratedsfrset`
Description: Set the `spheroidIntegratedSFR` property of an `standard` implementation of the `ageStatistics` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `agestatisticsstandardspheroidtimeweightedintegratedsfrcount`
Description: Return a count of the number of scalar properties in the `spheroidTimeWeightedIntegratedSFR` property of an `standard` implementation of the `ageStatistics` component class.
Code lines: 12
Contained by: module `galacticus_nodes`

function: `agestatisticsstandardspheroidtimeweightedintegratedsfrget`
Description: Get the `spheroidTimeWeightedIntegratedSFR` property of an `standard` implementation of the `ageStatistics` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: agestatisticsstandardspheroidtimeweightedintegratedsfrjcbnzr
Description: Indicate that the spheroidTimeWeightedIntegratedSFR property of an standard implementation of the ageStatistics component class is inactive for differential equation solving.
Code lines: 12
Contained by: module galacticus_nodes

subroutine: agestatisticsstandardspheroidtimeweightedintegratedsfrrate
Description: Accumulate to the rate of change of the spheroidTimeWeightedIntegratedSFR property of an standard implementation of the ageStatistics component class.
Code lines: 31
Contained by: module galacticus_nodes

function: agestatisticsstandardspheroidtimeweightedintegratedsfrrateget
Description: Get the rate of change of the spheroidTimeWeightedIntegratedSFR property of an standard implementation of the ageStatistics component class.
Code lines: 23
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: agestatisticsstandardspheroidtimeweightedintegratedsfrscale
Description: Set the absolute scale of the spheroidTimeWeightedIntegratedSFR property of an standard implementation of the ageStatistics component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: agestatisticsstandardspheroidtimeweightedintegratedsfrset
Description: Set the spheroidTimeWeightedIntegratedSFR property of an standard implementation of the ageStatistics component class.
Code lines: 10
Contained by: module galacticus_nodes

function: basicaccretionrate
Description: Returns the default value for the accretionRate property for the basic component class.
Code lines: 13
Contained by: module galacticus_nodes

function: basicaccretionrateattributematch
Description: Return a text list of component implementations in the basic class that have the desired attributes for the accretionRate property
Code lines: 63
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: basicaccretionratebertschinger
Description: Returns the default value for the accretionRateBertschinger property for the basic component class.
Code lines: 13
Contained by: module galacticus_nodes

function: `basicaccretionratebertschingerattributematch`
Description: Return a text list of component implementations in the `basic` class that have the desired attributes for the `accretionRateBertschinger` property
Code lines: 41
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `basicaccretionratebertschingerisgettable`
Description: Returns true if the `accretionRateBertschinger` property is gettable for the `basic` component class.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `basicaccretionratebertschingerrateget`
Description: Returns a zero rate for the `accretionRateBertschinger` property for the `basic` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `basicaccretionrateisgettable`
Description: Returns true if the `accretionRate` property is gettable for the `basic` component class.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `basicaccretionraterateget`
Description: Returns a zero rate for the `accretionRate` property for the `basic` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `basicextendedtrackingmassmaximumget`
Description: Get the `massMaximum` property of an `extendedTracking` implementation of the `basic` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `basicextendedtrackingmassmaximumset`
Description: Set the `massMaximum` property of an `extendedTracking` implementation of the `basic` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `basicextendedtrackingoutputcount`
Description: Increment the count of properties to output for a `extendedTracking` implementation of the `basic` component.
Code lines: 12
Contained by: module `galacticus_nodes`

subroutine: `basicextendedtrackingoutputnames`

Description: Return the names of properties to output for a `extendedTracking` implementation of the `basic` component.
Code lines: 17
Contained by: module `galacticus_nodes`

subroutine: `basicextendedtrackingpostoutput`
Description: Perform post-output processing for a `extendedTracking` implementation of the `basic` component.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `basicmass`
Description: Returns the default value for the `mass` property for the `basic` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `basicmassattributematch`
Description: Return a text list of component implementations in the `basic` class that have the desired attributes for the `mass` property
Code lines: 66
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `basicmassbertschinger`
Description: Returns the default value for the `massBertschinger` property for the `basic` component class.
Code lines: 17
Contained by: module `galacticus_nodes`

function: `basicmassbertschingerattributematch`
Description: Return a text list of component implementations in the `basic` class that have the desired attributes for the `massBertschinger` property
Code lines: 37
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `basicmassbertschingerisgettable`
Description: Returns true if the `massBertschinger` property is gettable for the `basic` component class.
Code lines: 9
Contained by: module `galacticus_nodes`

function: `basicmassbertschingerrateget`
Description: Returns a zero rate for the `massBertschinger` property for the `basic` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `basicmassisgettable`
Description: Returns true if the `mass` property is gettable for the `basic` component class.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `basicmassmaximum`
Description: Returns the default value for the `massMaximum` property for the `basic` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `basicmassmaximumattributematch`
Description: Return a text list of component implementations in the `basic` class that have the desired attributes for the `massMaximum` property
Code lines: 41
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `basicmassmaximumisgettable`
Description: Returns true if the `massMaximum` property is gettable for the `basic` component class.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `basicmassmaximumrateget`
Description: Returns a zero rate for the `massMaximum` property for the `basic` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `basicmassrateget`
Description: Returns a zero rate for the `mass` property for the `basic` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `basicmasstarget`
Description: Returns the default value for the `massTarget` property for the `basic` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `basicmasstargetattributematch`
Description: Return a text list of component implementations in the `basic` class that have the desired attributes for the `massTarget` property
Code lines: 63
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `basicmasstargetisgettable`
Description: Returns true if the `massTarget` property is gettable for the `basic` component class.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `basicmasstargetrateget`
Description: Returns a zero rate for the `massTarget` property for the `basic` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `basicnonevolvingmassget`
Description: Get the `mass` property of an `nonEvolving` implementation of the `basic` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `basicnonevolvingmassset`
Description: Set the `mass` property of an `nonEvolving` implementation of the `basic` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `basicnonevolvingoutputcount`
Description: Increment the count of properties to output for a `nonEvolving` implementation of the `basic` component.
Code lines: 21
Contained by: module `galacticus_nodes`

subroutine: `basicnonevolvingoutputnames`
Description: Return the names of properties to output for a `nonEvolving` implementation of the `basic` component.
Code lines: 43
Contained by: module `galacticus_nodes`

function: `basicnonevolvingtimecount`
Description: Return a count of the number of scalar properties in the `time` property of an `nonEvolving` implementation of the `basic` component class.
Code lines: 12
Contained by: module `galacticus_nodes`

function: `basicnonevolvingtimeget`
Description: Get the `time` property of an `nonEvolving` implementation of the `basic` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `basicnonevolvingtimejcbnzr`
Description: Indicate that the `time` property of an `nonEvolving` implementation of the `basic` component class is inactive for differential equation solving.
Code lines: 12
Contained by: module `galacticus_nodes`

function: `basicnonevolvingtimelastisolated`
Description: Return the `timeLastIsolated` property of the `Basic` component class.
Code lines: 16
Contained by: module `galacticus_nodes`

subroutine: `basicnonevolvingtimelastisolatedset`
Description: Set the `timeLastIsolated` property of an `nonEvolving` implementation of the `basic` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `basicnonevolvingtimerate`
Description: Accumulate to the rate of change of the `time` property of an `nonEvolving` implementation of the `basic` component class.
Code lines: 31
Contained by: module `galacticus_nodes`

function: `basicnonevolvingtimerateget`
Description: Get the rate of change of the `time` property of an `nonEvolving` implementation of the `basic` component class.
Code lines: 23
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `basicnonevolvingtimescale`
Description: Set the absolute scale of the `time` property of an `nonEvolving` implementation of the `basic` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

subroutine: `basicnonevolvingtimeset`
Description: Set the `time` property of an `nonEvolving` implementation of the `basic` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `basicnulloutputcount`
Description: Increment the count of properties to output for a `null` implementation of the `basic` component.
Code lines: 22
Contained by: module `galacticus_nodes`

subroutine: `basicnulloutputnames`
Description: Return the names of properties to output for a `null` implementation of the `basic` component.
Code lines: 39
Contained by: module `galacticus_nodes`

subroutine: `basicoutput`
Description: Populate output buffers with properties to output for a `basic` component.
Code lines: 46
Contained by: module `galacticus_nodes`
Modules used: `multi_counters`

subroutine: `basicoutputcount`
Description: Increment the count of properties to output for a generic `basic` component.
Code lines: 15
Contained by: module `galacticus_nodes`

subroutine: `basicoutputnames`
Description: Establish the names of properties to output for a generic `basic` component.

Code lines: 20
Contained by: module galacticus_nodes

subroutine: basicpostoutput
Description: Perform post-output processing of a basic component.
Code lines: 14
Contained by: module galacticus_nodes

function: basicradiusturnaround
Description: Returns the default value for the radiusTurnaround property for the basic component class.
Code lines: 17
Contained by: module galacticus_nodes

function: basicradiusturnaroundattributematch
Description: Return a text list of component implementations in the basic class that have the desired attributes for the radiusTurnaround property
Code lines: 37
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: basicradiusturnaroundgrowthrate
Description: Returns the default value for the radiusTurnaroundGrowthRate property for the basic component class.
Code lines: 17
Contained by: module galacticus_nodes

function: basicradiusturnaroundgrowthrateattributematch
Description: Return a text list of component implementations in the basic class that have the desired attributes for the radiusTurnaroundGrowthRate property
Code lines: 41
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: basicradiusturnaroundgrowthrateisgettable
Description: Returns true if the radiusTurnaroundGrowthRate property is gettable for the basic component class.
Code lines: 9
Contained by: module galacticus_nodes

function: basicradiusturnaroundgrowthraterateget
Description: Returns a zero rate for the radiusTurnaroundGrowthRate property for the basic component class.
Code lines: 13
Contained by: module galacticus_nodes

function: basicradiusturnaroundisgettable
Description: Returns true if the radiusTurnaround property is gettable for the basic component class.
Code lines: 9
Contained by: module galacticus_nodes

function: `basicradiusturnaroundrateget`
Description: Returns a zero rate for the `radiusTurnaround` property for the `basic` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `basicstandardaccretionrateget`
Description: Get the `accretionRate` property of an `standard` implementation of the `basic` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `basicstandardaccretionrateset`
Description: Set the `accretionRate` property of an `standard` implementation of the `basic` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `basicstandardextendedaccretionratebertschingerget`
Description: Get the `accretionRateBertschinger` property of an `standardExtended` implementation of the `basic` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `basicstandardextendedaccretionratebertschingerset`
Description: Set the `accretionRateBertschinger` property of an `standardExtended` implementation of the `basic` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `basicstandardextendedmassbertschingercount`
Description: Return a count of the number of scalar properties in the `massBertschinger` property of an `standardExtended` implementation of the `basic` component class.
Code lines: 12
Contained by: module `galacticus_nodes`

function: `basicstandardextendedmassbertschingerget`
Description: Get the value of the `massBertschinger` property of the `standardExtended` implementation of the `basic` component using a deferred function.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `basicstandardextendedmassbertschingergetfunction`
Description: Set the function to be used for the `get` method of the `massBertschinger` property of the `BasicStandardExtended` component.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `basicstandardextendedmassbertschingergetisattached`
Description: Return true if the deferred function used to get the `massBertschinger` property of the `BasicStandardExtended` component class has been attached.

Code lines: 8
Contained by: module galacticus_nodes

function: basicstandardextendedmassbertschingergetvalue
Description: Get the massBertschinger property of an standardExtended implementation of the basic component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: basicstandardextendedmassbertschingerjcbnzr
Description: Indicate that the massBertschinger property of an standardExtended implementation of the basic component class is inactive for differential equation solving.
Code lines: 12
Contained by: module galacticus_nodes

subroutine: basicstandardextendedmassbertschingerrate
Description: Accumulate to the rate of change of the massBertschinger property of an standardExtended implementation of the basic component class.
Code lines: 31
Contained by: module galacticus_nodes

function: basicstandardextendedmassbertschingerrateget
Description: Get the rate of change of the massBertschinger property of an standardExtended implementation of the basic component class.
Code lines: 23
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: basicstandardextendedmassbertschingerscale
Description: Set the absolute scale of the massBertschinger property of an standardExtended implementation of the basic component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: basicstandardextendedmassbertschingerset
Description: Set the massBertschinger property of an standardExtended implementation of the basic component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: basicstandardextendedoutputcount
Description: Increment the count of properties to output for a standardExtended implementation of the basic component.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: basicstandardextendedoutputnames
Description: Return the names of properties to output for a standardExtended implementation of the basic component.
Code lines: 23
Contained by: module galacticus_nodes

subroutine: `basicstandardextendedpostoutput`
Description: Perform post-output processing for a `standardExtended` implementation of the `basic` component.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `basicstandardextendedradiusturnaroundcount`
Description: Return a count of the number of scalar properties in the `radiusTurnaround` property of an `standardExtended` implementation of the `basic` component class.
Code lines: 12
Contained by: module `galacticus_nodes`

function: `basicstandardextendedradiusturnaroundget`
Description: Get the value of the `radiusTurnaround` property of the `standardExtended` implementation of the `basic` component using a deferred function.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `basicstandardextendedradiusturnaroundgetfunction`
Description: Set the function to be used for the `get` method of the `radiusTurnaround` property of the `BasicStandardExtended` component.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `basicstandardextendedradiusturnaroundgetisattached`
Description: Return true if the deferred function used to get the `radiusTurnaround` property of the `BasicStandardExtended` component class has been attached.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `basicstandardextendedradiusturnaroundgetvalue`
Description: Get the `radiusTurnaround` property of an `standardExtended` implementation of the `basic` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `basicstandardextendedradiusturnaroundgrowthrateget`
Description: Get the `radiusTurnaroundGrowthRate` property of an `standardExtended` implementation of the `basic` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `basicstandardextendedradiusturnaroundgrowthrateset`
Description: Set the `radiusTurnaroundGrowthRate` property of an `standardExtended` implementation of the `basic` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `basicstandardextendedradiusturnaroundjcbnzr`

Description: Indicate that the `radiusTurnaround` property of an `standardExtended` implementation of the `basic` component class is inactive for differential equation solving.
Code lines: 12
Contained by: module galacticus_nodes

subroutine: `basicstandardextendedradiusturnaroundrate`
Description: Accumulate to the rate of change of the `radiusTurnaround` property of an `standardExtended` implementation of the `basic` component class.
Code lines: 31
Contained by: module galacticus_nodes

function: `basicstandardextendedradiusturnaroundrateget`
Description: Get the rate of change of the `radiusTurnaround` property of an `standardExtended` implementation of the `basic` component class.
Code lines: 23
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: `basicstandardextendedradiusturnaroundscale`
Description: Set the absolute scale of the `radiusTurnaround` property of an `standardExtended` implementation of the `basic` component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: `basicstandardextendedradiusturnaroundset`
Description: Set the `radiusTurnaround` property of an `standardExtended` implementation of the `basic` component class.
Code lines: 10
Contained by: module galacticus_nodes

function: `basicstandardmasscount`
Description: Return a count of the number of scalar properties in the `mass` property of an `standard` implementation of the `basic` component class.
Code lines: 12
Contained by: module galacticus_nodes

function: `basicstandardmassget`
Description: Get the `mass` property of an `standard` implementation of the `basic` component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: `basicstandardmassjcbnzr`
Description: Indicate that the `mass` property of an `standard` implementation of the `basic` component class is inactive for differential equation solving.
Code lines: 12
Contained by: module galacticus_nodes

subroutine: `basicstandardmassrate`
Description: Accumulate to the rate of change of the `mass` property of an `standard` implementation of the `basic` component class.
Code lines: 31

Contained by: module galacticus_nodes

function: basicstandardmassrateget
Description: Get the rate of change of the mass property of an standard implementation of the basic component class.
Code lines: 23
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: basicstandardmassscale
Description: Set the absolute scale of the mass property of an standard implementation of the basic component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: basicstandardmassset
Description: Set the mass property of an standard implementation of the basic component class.
Code lines: 10
Contained by: module galacticus_nodes

function: basicstandardmasstargetget
Description: Get the massTarget property of an standard implementation of the basic component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: basicstandardmasstargetset
Description: Set the massTarget property of an standard implementation of the basic component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: basicstandardoutputcount
Description: Increment the count of properties to output for a standard implementation of the basic component.
Code lines: 21
Contained by: module galacticus_nodes

subroutine: basicstandardoutputnames
Description: Return the names of properties to output for a standard implementation of the basic component.
Code lines: 43
Contained by: module galacticus_nodes

function: basicstandardtimecount
Description: Return a count of the number of scalar properties in the time property of an standard implementation of the basic component class.
Code lines: 12
Contained by: module galacticus_nodes

function: basicstandardtimeget
Description: Get the time property of an standard implementation of the basic component class.

Code lines: 10
Contained by: module galacticus_nodes

subroutine: basicstandardtimejcbnzr
Description: Indicate that the time property of an standard implementation of the basic component class is inactive for differential equation solving.
Code lines: 12
Contained by: module galacticus_nodes

function: basicstandardtimelastisolated
Description: Return the timeLastIsolated property of the Basic component class.
Code lines: 16
Contained by: module galacticus_nodes

subroutine: basicstandardtimelastisolatedset
Description: Set the timeLastIsolated property of an standard implementation of the basic component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: basicstandardtimerate
Description: Accumulate to the rate of change of the time property of an standard implementation of the basic component class.
Code lines: 31
Contained by: module galacticus_nodes

function: basicstandardtimerateget
Description: Get the rate of change of the time property of an standard implementation of the basic component class.
Code lines: 23
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: basicstandardtimescale
Description: Set the absolute scale of the time property of an standard implementation of the basic component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: basicstandardtimeset
Description: Set the time property of an standard implementation of the basic component class.
Code lines: 10
Contained by: module galacticus_nodes

function: basicstandardtrackingmassmaximumget
Description: Get the massMaximum property of an standardTracking implementation of the basic component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: basicstandardtrackingmassmaximumset

Description: Set the `massMaximum` property of an `standardTracking` implementation of the `basic` component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: `basicstandardtrackingoutputcount`
Description: Increment the count of properties to output for a `standardTracking` implementation of the `basic` component.
Code lines: 12
Contained by: module galacticus_nodes

subroutine: `basicstandardtrackingoutputnames`
Description: Return the names of properties to output for a `standardTracking` implementation of the `basic` component.
Code lines: 17
Contained by: module galacticus_nodes

subroutine: `basicstandardtrackingpostoutput`
Description: Perform post-output processing for a `standardTracking` implementation of the `basic` component.
Code lines: 10
Contained by: module galacticus_nodes

function: `basictime`
Description: Returns the default value for the `time` property for the `basic` component class.
Code lines: 13
Contained by: module galacticus_nodes

function: `basictimeattributematch`
Description: Return a text list of component implementations in the `basic` class that have the desired attributes for the `time` property
Code lines: 64
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: `basictimeisgettable`
Description: Returns true if the `time` property is gettable for the `basic` component class.
Code lines: 8
Contained by: module galacticus_nodes

function: `basictimelastisolated`
Description: Returns the default value for the `timeLastIsolated` property for the `basic` component class.
Code lines: 13
Contained by: module galacticus_nodes

function: `basictimelastisolatedattributematch`
Description: Return a text list of component implementations in the `basic` class that have the desired attributes for the `timeLastIsolated` property
Code lines: 74
Contained by: module galacticus_nodes

Modules used: iso_varying_string

function: basictimelastisolatedisgettable
Description: Returns true if the timeLastIsolated property is gettable for the basic component class.
Code lines: 8
Contained by: module galacticus_nodes

function: basictimelastisolatedrateget
Description: Returns a zero rate for the timeLastIsolated property for the basic component class.
Code lines: 13
Contained by: module galacticus_nodes

function: basictimerateget
Description: Returns a zero rate for the time property for the basic component class.
Code lines: 13
Contained by: module galacticus_nodes

function: blackholeaccretionrate
Description: Returns the default value for the accretionRate property for the blackHole component class.
Code lines: 13
Contained by: module galacticus_nodes

function: blackholeaccretionrateattributematch
Description: Return a text list of component implementations in the blackHole class that have the desired attributes for the accretionRate property
Code lines: 41
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: blackholeaccretionrateisgettable
Description: Returns true if the accretionRate property is gettable for the blackHole component class.
Code lines: 8
Contained by: module galacticus_nodes

function: blackholeaccretionraterateget
Description: Returns a zero rate for the accretionRate property for the blackHole component class.
Code lines: 13
Contained by: module galacticus_nodes

function: blackholemass
Description: Returns the default value for the mass property for the blackHole component class.
Code lines: 21
Contained by: module galacticus_nodes

function: blackholemassattributematch
Description: Return a text list of component implementations in the blackHole class that have the desired attributes for the mass property
Code lines: 46
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: `blackholemassisgettable`
Description: Returns true if the `mass` property is gettable for the `blackHole` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `blackholemassrateget`
Description: Returns a zero rate for the `mass` property for the `blackHole` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `blackholemassseed`
Description: Returns the default value for the `massSeed` property for the `blackHole` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `blackholemassseedattributematch`
Description: Return a text list of component implementations in the `blackHole` class that have the desired attributes for the `massSeed` property
Code lines: 52
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `blackholemassseedisgettable`
Description: Returns true if the `massSeed` property is gettable for the `blackHole` component class.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `blackholemassseedrateget`
Description: Returns a zero rate for the `massSeed` property for the `blackHole` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

subroutine: `blackholenoncentraloutputcount`
Description: Increment the count of properties to output for a `nonCentral` implementation of the `blackHole` component.
Code lines: 13
Contained by: module `galacticus_nodes`

subroutine: `blackholenoncentraloutputnames`
Description: Return the names of properties to output for a `nonCentral` implementation of the `blackHole` component.
Code lines: 18
Contained by: module `galacticus_nodes`

subroutine: `blackholenoncentralpostoutput`
Description: Perform post-output processing for a `nonCentral` implementation of the `blackHole` component.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `blackholenoncentralradialpositioncount`
Description: Return a count of the number of scalar properties in the `radialPosition` property of an `nonCentral` implementation of the `blackHole` component class.
Code lines: 12
Contained by: module `galacticus_nodes`

function: `blackholenoncentralradialpositionget`
Description: Get the `radialPosition` property of an `nonCentral` implementation of the `blackHole` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `blackholenoncentralradialpositionjcbnzr`
Description: Indicate that the `radialPosition` property of an `nonCentral` implementation of the `blackHole` component class is inactive for differential equation solving.
Code lines: 12
Contained by: module `galacticus_nodes`

subroutine: `blackholenoncentralradialpositionrate`
Description: Accumulate to the rate of change of the `radialPosition` property of an `nonCentral` implementation of the `blackHole` component class.
Code lines: 31
Contained by: module `galacticus_nodes`

function: `blackholenoncentralradialpositionrateget`
Description: Get the rate of change of the `radialPosition` property of an `nonCentral` implementation of the `blackHole` component class.
Code lines: 23
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `blackholenoncentralradialpositionscale`
Description: Set the absolute scale of the `radialPosition` property of an `nonCentral` implementation of the `blackHole` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

subroutine: `blackholenoncentralradialpositionset`
Description: Set the `radialPosition` property of an `nonCentral` implementation of the `blackHole` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `blackholenulloutputcount`
Description: Increment the count of properties to output for a `null` implementation of the `blackHole` component.
Code lines: 22
Contained by: module `galacticus_nodes`

subroutine: `blackholenulloutputnames`
Description: Return the names of properties to output for a `null` implementation of the `blackHole` component.
Code lines: 39
Contained by: module `galacticus_nodes`

subroutine: `blackholeoutput`
Description: Populate output buffers with properties to output for a `blackHole` component.
Code lines: 38
Contained by: module `galacticus_nodes`
Modules used: `multi_counters`

subroutine: `blackholeoutputcount`
Description: Increment the count of properties to output for a generic `blackHole` component.
Code lines: 15
Contained by: module `galacticus_nodes`

subroutine: `blackholeoutputnames`
Description: Establish the names of properties to output for a generic `blackHole` component.
Code lines: 20
Contained by: module `galacticus_nodes`

subroutine: `blackholepostoutput`
Description: Perform post-output processing of a `blackHole` component.
Code lines: 14
Contained by: module `galacticus_nodes`

function: `blackholeradialposition`
Description: Returns the default value for the `radialPosition` property for the `blackHole` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `blackholeradialpositionattributematch`
Description: Return a text list of component implementations in the `blackHole` class that have the desired attributes for the `radialPosition` property
Code lines: 41
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `blackholeradialpositionisgettable`
Description: Returns true if the `radialPosition` property is gettable for the `blackHole` component class.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `blackholeradialpositionrateget`
Description: Returns a zero rate for the `radialPosition` property for the `blackHole` component class.
Code lines: 13

Contained by: module galacticus_nodes

function: blackholeradiativeefficiency
Description: Returns the default value for the radiativeEfficiency property for the blackHole component class.
Code lines: 13
Contained by: module galacticus_nodes

function: blackholeradiativeefficiencyattributematch
Description: Return a text list of component implementations in the blackHole class that have the desired attributes for the radiativeEfficiency property
Code lines: 41
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: blackholeradiativeefficiencyisgettable
Description: Returns true if the radiativeEfficiency property is gettable for the blackHole component class.
Code lines: 8
Contained by: module galacticus_nodes

function: blackholeradiativeefficiencyrateget
Description: Returns a zero rate for the radiativeEfficiency property for the blackHole component class.
Code lines: 13
Contained by: module galacticus_nodes

function: blackholesimplemasscount
Description: Return a count of the number of scalar properties in the mass property of an simple implementation of the blackHole component class.
Code lines: 12
Contained by: module galacticus_nodes

function: blackholesimplemassget
Description: Get the mass property of an simple implementation of the blackHole component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: blackholesimplemassjcbnzr
Description: Indicate that the mass property of an simple implementation of the blackHole component class is inactive for differential equation solving.
Code lines: 12
Contained by: module galacticus_nodes

subroutine: blackholesimplemassrate
Description: Accumulate to the rate of change of the mass property of an simple implementation of the blackHole component class.
Code lines: 31
Contained by: module galacticus_nodes

function: blackholesimplemassrateget

Description: Get the rate of change of the `mass` property of an `simple` implementation of the `blackHole` component class.
Code lines: 23
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `blackholesimplemassscale`
Description: Set the absolute scale of the `mass` property of an `simple` implementation of the `blackHole` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

subroutine: `blackholesimplemassset`
Description: Set the `mass` property of an `simple` implementation of the `blackHole` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `blackholesimpleoutputcount`
Description: Increment the count of properties to output for a `simple` implementation of the `blackHole` component.
Code lines: 21
Contained by: module `galacticus_nodes`

subroutine: `blackholesimpleoutputnames`
Description: Return the names of properties to output for a `simple` implementation of the `blackHole` component.
Code lines: 37
Contained by: module `galacticus_nodes`

function: `blackholespin`
Description: Returns the default value for the `spin` property for the `blackHole` component class.
Code lines: 17
Contained by: module `galacticus_nodes`

function: `blackholespinattributematch`
Description: Return a text list of component implementations in the `blackHole` class that have the desired attributes for the `spin` property
Code lines: 37
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `blackholespinisgettable`
Description: Returns true if the `spin` property is gettable for the `blackHole` component class.
Code lines: 9
Contained by: module `galacticus_nodes`

function: `blackholespinrateget`
Description: Returns a zero rate for the `spin` property for the `blackHole` component class.
Code lines: 13

Contained by: module galacticus_nodes

function: blackholespinseed
Description: Returns the default value for the spinSeed property for the blackHole component class.
Code lines: 13
Contained by: module galacticus_nodes

function: blackholespinseedattributematch
Description: Return a text list of component implementations in the blackHole class that have the desired attributes for the spinSeed property
Code lines: 41
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: blackholespinseedisgettable
Description: Returns true if the spinSeed property is gettable for the blackHole component class.
Code lines: 8
Contained by: module galacticus_nodes

function: blackholespinseedrateget
Description: Returns a zero rate for the spinSeed property for the blackHole component class.
Code lines: 13
Contained by: module galacticus_nodes

function: blackholestandardaccretionrateget
Description: Get the value of the accretionRate property of the standard implementation of the blackHole component using a deferred function.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: blackholestandardaccretionrategetfunction
Description: Set the function to be used for the get method of the accretionRate property of the BlackHoleStandard component.
Code lines: 10
Contained by: module galacticus_nodes

function: blackholestandardaccretionrategetisattached
Description: Return true if the deferred function used to get the accretionRate property of the BlackHoleStandard component class has been attached.
Code lines: 8
Contained by: module galacticus_nodes

function: blackholestandardmasscount
Description: Return a count of the number of scalar properties in the mass property of an standard implementation of the blackHole component class.
Code lines: 12
Contained by: module galacticus_nodes

function: blackholestandardmassget

Description: Get the `mass` property of an `standard` implementation of the `blackHole` component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: `blackholestandardmassjcbnzr`
Description: Indicate that the `mass` property of an `standard` implementation of the `blackHole` component class is inactive for differential equation solving.
Code lines: 12
Contained by: module galacticus_nodes

subroutine: `blackholestandardmassrate`
Description: Accumulate to the rate of change of the `mass` property of an `standard` implementation of the `blackHole` component class.
Code lines: 31
Contained by: module galacticus_nodes

function: `blackholestandardmassrateget`
Description: Get the rate of change of the `mass` property of an `standard` implementation of the `blackHole` component class.
Code lines: 23
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: `blackholestandardmassscale`
Description: Set the absolute scale of the `mass` property of an `standard` implementation of the `blackHole` component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: `blackholestandardmassset`
Description: Set the `mass` property of an `standard` implementation of the `blackHole` component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: `blackholestandardoutputcount`
Description: Increment the count of properties to output for a `standard` implementation of the `blackHole` component.
Code lines: 20
Contained by: module galacticus_nodes

subroutine: `blackholestandardoutputnames`
Description: Return the names of properties to output for a `standard` implementation of the `blackHole` component.
Code lines: 42
Contained by: module galacticus_nodes

function: `blackholestandardradialpositionget`
Description: Get the `radialPosition` property of an `standard` implementation of the `blackHole` component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: `blackholestandardradialpositionset`
Description: Set the `radialPosition` property of an `standard` implementation of the `blackHole` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `blackholestandardradiativeefficiencyget`
Description: Get the value of the `radiativeEfficiency` property of the `standard` implementation of the `blackHole` component using a deferred function.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `blackholestandardradiativeefficiencygetfunction`
Description: Set the function to be used for the `get` method of the `radiativeEfficiency` property of the `BlackHoleStandard` component.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `blackholestandardradiativeefficiencygetisattached`
Description: Return true if the deferred function used to get the `radiativeEfficiency` property of the `BlackHoleStandard` component class has been attached.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `blackholestandardspincount`
Description: Return a count of the number of scalar properties in the `spin` property of an `standard` implementation of the `blackHole` component class.
Code lines: 12
Contained by: module `galacticus_nodes`

subroutine: `blackholestandardspinjcbnzr`
Description: Indicate that the `spin` property of an `standard` implementation of the `blackHole` component class is inactive for differential equation solving.
Code lines: 12
Contained by: module `galacticus_nodes`

subroutine: `blackholestandardspinrate`
Description: Accumulate to the rate of change of the `spin` property of an `standard` implementation of the `blackHole` component class.
Code lines: 31
Contained by: module `galacticus_nodes`

function: `blackholestandardspinrateget`
Description: Get the rate of change of the `spin` property of an `standard` implementation of the `blackHole` component class.
Code lines: 23
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `blackholestandardspinscale`
Description: Set the absolute scale of the `spin` property of an `standard` implementation of the `blackHole` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

subroutine: `blackholestandardspinset`
Description: Set the `spin` property of an `standard` implementation of the `blackHole` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `blackholestandardtripleinteractiontimeget`
Description: Get the `tripleInteractionTime` property of an `standard` implementation of the `blackHole` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `blackholestandardtripleinteractiontimeset`
Description: Set the `tripleInteractionTime` property of an `standard` implementation of the `blackHole` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `blackholetripleinteractiontime`
Description: Returns the default value for the `tripleInteractionTime` property for the `blackHole` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `blackholetripleinteractiontimeattributematch`
Description: Return a text list of component implementations in the `blackHole` class that have the desired attributes for the `tripleInteractionTime` property
Code lines: 41
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `blackholetripleinteractiontimeisgettable`
Description: Returns true if the `tripleInteractionTime` property is gettable for the `blackHole` component class.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `blackholetripleinteractiontimerateget`
Description: Returns a zero rate for the `tripleInteractionTime` property for the `blackHole` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `boolean_false`

Description: Returns Boolean false always.
Code lines: 6
Contained by: module galacticus_nodes

function: boolean_true
Description: Returns Boolean true always.
Code lines: 6
Contained by: module galacticus_nodes

subroutine: darkmatterprofilenulloutputcount
Description: Increment the count of properties to output for a null implementation of the darkMatterProfile component.
Code lines: 22
Contained by: module galacticus_nodes

subroutine: darkmatterprofilenulloutputnames
Description: Return the names of properties to output for a null implementation of the darkMatterProfile component.
Code lines: 39
Contained by: module galacticus_nodes

subroutine: darkmatterprofileoutput
Description: Populate output buffers with properties to output for a darkMatterProfile component.
Code lines: 42
Contained by: module galacticus_nodes
Modules used: multi_counters

subroutine: darkmatterprofileoutputcount
Description: Increment the count of properties to output for a generic darkMatterProfile component.
Code lines: 15
Contained by: module galacticus_nodes

subroutine: darkmatterprofileoutputnames
Description: Establish the names of properties to output for a generic darkMatterProfile component.
Code lines: 20
Contained by: module galacticus_nodes

subroutine: darkmatterprofilepostoutput
Description: Perform post-output processing of a darkMatterProfile component.
Code lines: 14
Contained by: module galacticus_nodes

function: darkmatterprofilescale
Description: Returns the default value for the scale property for the darkMatterProfile component class.
Code lines: 21
Contained by: module galacticus_nodes

function: darkmatterprofilescaleattributematch

Description: Return a text list of component implementations in the `darkMatterProfile` class that have the desired attributes for the `scale` property
Code lines: 46
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `darkmatterprofilescalegrowthrate`
Description: Returns the default value for the `scaleGrowthRate` property for the `darkMatterProfile` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `darkmatterprofilescalegrowthrateattributematch`
Description: Return a text list of component implementations in the `darkMatterProfile` class that have the desired attributes for the `scaleGrowthRate` property
Code lines: 52
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `darkmatterprofilescalegrowthrateisgettable`
Description: Returns true if the `scaleGrowthRate` property is gettable for the `darkMatterProfile` component class.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `darkmatterprofilescalegrowthraterateget`
Description: Returns a zero rate for the `scaleGrowthRate` property for the `darkMatterProfile` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `darkmatterprofilescaleisgettable`
Description: Returns true if the `scale` property is gettable for the `darkMatterProfile` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `darkmatterprofilescaleislimited`
Description: Returns the default value for the `scaleIsLimited` property for the `darkMatterProfile` component class.
Code lines: 17
Contained by: module `galacticus_nodes`

function: `darkmatterprofilescaleislimitedattributematch`
Description: Return a text list of component implementations in the `darkMatterProfile` class that have the desired attributes for the `scaleIsLimited` property
Code lines: 41
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `darkmatterprofilescaleislimitedisgettable`

Description: Returns true if the `scaleIsLimited` property is gettable for the `darkMatterProfile` component class.
Code lines: 9
Contained by: module `galacticus_nodes`

function: `darkmatterprofilescaleislimitedrateget`
Description: Returns a zero rate for the `scaleIsLimited` property for the `darkMatterProfile` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

subroutine: `darkmatterprofilescaleoutputcount`
Description: Increment the count of properties to output for a `scale` implementation of the `darkMatterProfile` component.
Code lines: 21
Contained by: module `galacticus_nodes`

subroutine: `darkmatterprofilescaleoutputnames`
Description: Return the names of properties to output for a `scale` implementation of the `darkMatterProfile` component.
Code lines: 37
Contained by: module `galacticus_nodes`

subroutine: `darkmatterprofilescalepresetoutputcount`
Description: Increment the count of properties to output for a `scalePreset` implementation of the `darkMatterProfile` component.
Code lines: 21
Contained by: module `galacticus_nodes`

subroutine: `darkmatterprofilescalepresetoutputnames`
Description: Return the names of properties to output for a `scalePreset` implementation of the `darkMatterProfile` component.
Code lines: 37
Contained by: module `galacticus_nodes`

function: `darkmatterprofilescalepresetscalecount`
Description: Return a count of the number of scalar properties in the `scale` property of an `scalePreset` implementation of the `darkMatterProfile` component class.
Code lines: 12
Contained by: module `galacticus_nodes`

function: `darkmatterprofilescalepresetscaleget`
Description: Get the `scale` property of an `scalePreset` implementation of the `darkMatterProfile` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `darkmatterprofilescalepresetscalegrowthrateget`
Description: Get the `scaleGrowthRate` property of an `scalePreset` implementation of the `darkMatterProfile` component class.
Code lines: 10

Contained by: module galacticus_nodes

subroutine: darkmatterprofilescalepresetscalegrowthrateset
Description: Set the scaleGrowthRate property of an scalePreset implementation of the darkMatterProfile component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: darkmatterprofilescalepresetscalejcbnzr
Description: Indicate that the scale property of an scalePreset implementation of the darkMatterProfile component class is inactive for differential equation solving.
Code lines: 12
Contained by: module galacticus_nodes

subroutine: darkmatterprofilescalepresetscalerate
Description: Accumulate to the rate of change of the scale property of an scalePreset implementation of the darkMatterProfile component class.
Code lines: 31
Contained by: module galacticus_nodes

function: darkmatterprofilescalepresetscalerateget
Description: Get the rate of change of the scale property of an scalePreset implementation of the darkMatterProfile component class.
Code lines: 23
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: darkmatterprofilescalepresetscalescale
Description: Set the absolute scale of the scale property of an scalePreset implementation of the darkMatterProfile component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: darkmatterprofilescalepresetscaleset
Description: Set the scale property of an scalePreset implementation of the darkMatterProfile component class.
Code lines: 10
Contained by: module galacticus_nodes

function: darkmatterprofilescalerateget
Description: Returns a zero rate for the scale property for the darkMatterProfile component class.
Code lines: 13
Contained by: module galacticus_nodes

function: darkmatterprofilescalescalecount
Description: Return a count of the number of scalar properties in the scale property of an scale implementation of the darkMatterProfile component class.
Code lines: 12
Contained by: module galacticus_nodes

function: `darkmatterprofilescalescaleget`
Description: Get the value of the `scale` property of the `scale` implementation of the `darkMatterProfile` component using a deferred function.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `darkmatterprofilescalescalegetfunction`
Description: Set the function to be used for the `get` method of the `scale` property of the `DarkMatterProfileScale` component.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `darkmatterprofilescalescalegetisattached`
Description: Return true if the deferred function used to get the `scale` property of the `DarkMatterProfileScale` component class has been attached.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `darkmatterprofilescalescalegetvalue`
Description: Get the `scale` property of an `scale` implementation of the `darkMatterProfile` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `darkmatterprofilescalescalegrowthrateget`
Description: Get the `scaleGrowthRate` property of an `scale` implementation of the `darkMatterProfile` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `darkmatterprofilescalescalegrowthrateset`
Description: Set the `scaleGrowthRate` property of an `scale` implementation of the `darkMatterProfile` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `darkmatterprofilescalescaleislimitedget`
Description: Get the `scaleIsLimited` property of an `scale` implementation of the `darkMatterProfile` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `darkmatterprofilescalescaleislimitedset`
Description: Set the `scaleIsLimited` property of an `scale` implementation of the `darkMatterProfile` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `darkmatterprofilescalescalejcbnzr`

Description: Indicate that the `scale` property of an `scale` implementation of the `darkMatterProfile` component class is inactive for differential equation solving.
Code lines: 12
Contained by: module `galacticus_nodes`

subroutine: `darkmatterprofilescalescalerate`
Description: Accumulate to the rate of change of the `scale` property of an `scale` implementation of the `darkMatterProfile` component class.
Code lines: 31
Contained by: module `galacticus_nodes`

function: `darkmatterprofilescalescalerateget`
Description: Get the rate of change of the `scale` property of an `scale` implementation of the `darkMatterProfile` component class.
Code lines: 23
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `darkmatterprofilescalescalescale`
Description: Set the absolute scale of the `scale` property of an `scale` implementation of the `darkMatterProfile` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

subroutine: `darkmatterprofilescalescaleset`
Description: Set the `scale` property of an `scale` implementation of the `darkMatterProfile` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `darkmatterprofilescaleshapeoutputcount`
Description: Increment the count of properties to output for a `scaleShape` implementation of the `darkMatterProfile` component.
Code lines: 13
Contained by: module `galacticus_nodes`

subroutine: `darkmatterprofilescaleshapeoutputnames`
Description: Return the names of properties to output for a `scaleShape` implementation of the `darkMatterProfile` component.
Code lines: 23
Contained by: module `galacticus_nodes`

subroutine: `darkmatterprofilescaleshapepostoutput`
Description: Perform post-output processing for a `scaleShape` implementation of the `darkMatterProfile` component.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `darkmatterprofilescaleshapeshapecount`
Description: Return a count of the number of scalar properties in the `shape` property of an `scaleShape` implementation of the `darkMatterProfile` component class.

Code lines: 12
Contained by: module galacticus_nodes

function: darkmatterprofilescaleshapeshapeget
Description: Get the value of the shape property of the scaleShape implementation of the darkMatterProfile component using a deferred function.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: darkmatterprofilescaleshapeshapegetfunction
Description: Set the function to be used for the get method of the shape property of the DarkMatterProfileScaleShape component.
Code lines: 10
Contained by: module galacticus_nodes

function: darkmatterprofilescaleshapeshapegetisattached
Description: Return true if the deferred function used to get the shape property of the DarkMatterProfileScaleShape component class has been attached.
Code lines: 8
Contained by: module galacticus_nodes

function: darkmatterprofilescaleshapeshapegetvalue
Description: Get the shape property of an scaleShape implementation of the darkMatterProfile component class.
Code lines: 10
Contained by: module galacticus_nodes

function: darkmatterprofilescaleshapeshapegrowthrateget
Description: Get the shapeGrowthRate property of an scaleShape implementation of the darkMatterProfile component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: darkmatterprofilescaleshapeshapegrowthrateset
Description: Set the shapeGrowthRate property of an scaleShape implementation of the darkMatterProfile component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: darkmatterprofilescaleshapeshapejcbnzr
Description: Indicate that the shape property of an scaleShape implementation of the darkMatterProfile component class is inactive for differential equation solving.
Code lines: 12
Contained by: module galacticus_nodes

subroutine: darkmatterprofilescaleshapeshaperate
Description: Accumulate to the rate of change of the shape property of an scaleShape implementation of the darkMatterProfile component class.
Code lines: 31
Contained by: module galacticus_nodes

function: `darkmatterprofilescaleshapeshaperateget`
Description: Get the rate of change of the `shape` property of an `scaleShape` implementation of the `darkMatterProfile` component class.
Code lines: 23
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `darkmatterprofilescaleshapeshapescale`
Description: Set the absolute scale of the `shape` property of an `scaleShape` implementation of the `darkMatterProfile` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

subroutine: `darkmatterprofilescaleshapeshapeset`
Description: Set the `shape` property of an `scaleShape` implementation of the `darkMatterProfile` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `darkmatterprofileshape`
Description: Returns the default value for the `shape` property for the `darkMatterProfile` component class.
Code lines: 17
Contained by: module `galacticus_nodes`

function: `darkmatterprofileshapeattributematch`
Description: Return a text list of component implementations in the `darkMatterProfile` class that have the desired attributes for the `shape` property
Code lines: 28
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `darkmatterprofileshapegrowthrate`
Description: Returns the default value for the `shapeGrowthRate` property for the `darkMatterProfile` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `darkmatterprofileshapegrowthrateattributematch`
Description: Return a text list of component implementations in the `darkMatterProfile` class that have the desired attributes for the `shapeGrowthRate` property
Code lines: 30
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `darkmatterprofileshapegrowthrateisgettable`
Description: Returns true if the `shapeGrowthRate` property is gettable for the `darkMatterProfile` component class.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `darkmatterprofileshapegrowthraterateget`
Description: Returns a zero rate for the `shapeGrowthRate` property for the `darkMatterProfile` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `darkmatterprofileshapeisgettable`
Description: Returns true if the `shape` property is gettable for the `darkMatterProfile` component class.
Code lines: 9
Contained by: module `galacticus_nodes`

function: `darkmatterprofileshaperateget`
Description: Returns a zero rate for the `shape` property for the `darkMatterProfile` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `diskabundancesgas`
Description: Returns the default value for the `abundancesGas` property for the `disk` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `diskabundancesgasattributematch`
Description: Return a text list of component implementations in the `disk` class that have the desired attributes for the `abundancesGas` property
Code lines: 46
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `diskabundancesgasisgettable`
Description: Returns true if the `abundancesGas` property is gettable for the `disk` component class.
Code lines: 8
Contained by: module `galacticus_nodes`

subroutine: `diskabundancesgasrate`
Description: Accept a rate set for the `abundancesGas` property of the `disk` component class. Trigger an interrupt to create the component.
Code lines: 21
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

function: `diskabundancesgasrateget`
Description: Returns a zero rate for the `abundancesGas` property for the `disk` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `diskabundancesstellar`
Description: Returns the default value for the `abundancesStellar` property for the `disk` component class.
Code lines: 13

Contained by: module galacticus_nodes

function: diskabundancesstellarattributematch
Description: Return a text list of component implementations in the disk class that have the desired attributes for the abundancesStellar property
Code lines: 46
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: diskabundancesstellarisgettable
Description: Returns true if the abundancesStellar property is gettable for the disk component class.
Code lines: 8
Contained by: module galacticus_nodes

function: diskabundancesstellarrateget
Description: Returns a zero rate for the abundancesStellar property for the disk component class.
Code lines: 13
Contained by: module galacticus_nodes

function: diskangularmomentum
Description: Returns the default value for the angularMomentum property for the disk component class.
Code lines: 13
Contained by: module galacticus_nodes

function: diskangularmomentumattributematch
Description: Return a text list of component implementations in the disk class that have the desired attributes for the angularMomentum property
Code lines: 28
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: diskangularmomentumisgettable
Description: Returns true if the angularMomentum property is gettable for the disk component class.
Code lines: 8
Contained by: module galacticus_nodes

subroutine: diskangularmomentumrate
Description: Accept a rate set for the angularMomentum property of the disk component class. Trigger an interrupt to create the component.
Code lines: 21
Contained by: module galacticus_nodes
Modules used: galacticus_error

function: diskangularmomentumrateget
Description: Returns a zero rate for the angularMomentum property for the disk component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: diskcreatebyinterrupt

Description: Create the `disk` component of `self` via an interrupt.
Code lines: 10
Contained by: module galacticus_nodes

function: `diskfractionmassretained`
Description: Returns the default value for the `fractionMassRetained` property for the `disk` component class.
Code lines: 13
Contained by: module galacticus_nodes

function: `diskfractionmassretainedattributematch`
Description: Return a text list of component implementations in the `disk` class that have the desired attributes for the `fractionMassRetained` property
Code lines: 28
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: `diskfractionmassretainedisgettable`
Description: Returns true if the `fractionMassRetained` property is gettable for the `disk` component class.
Code lines: 8
Contained by: module galacticus_nodes

function: `diskfractionmassretainedrateget`
Description: Returns a zero rate for the `fractionMassRetained` property for the `disk` component class.
Code lines: 13
Contained by: module galacticus_nodes

function: `diskhalfmassradius`
Description: Returns the default value for the `halfMassRadius` property for the `disk` component class.
Code lines: 13
Contained by: module galacticus_nodes

function: `diskhalfmassradiusattributematch`
Description: Return a text list of component implementations in the `disk` class that have the desired attributes for the `halfMassRadius` property
Code lines: 41
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: `diskhalfmassradiusisgettable`
Description: Returns true if the `halfMassRadius` property is gettable for the `disk` component class.
Code lines: 8
Contained by: module galacticus_nodes

function: `diskhalfmassradiusrateget`
Description: Returns a zero rate for the `halfMassRadius` property for the `disk` component class.
Code lines: 13
Contained by: module galacticus_nodes

function: `diskisinitialized`
Description: Returns the default value for the `isInitialized` property for the `disk` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `diskisinitializedattributematch`
Description: Return a text list of component implementations in the `disk` class that have the desired attributes for the `isInitialized` property
Code lines: 52
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `diskisinitializedisgettable`
Description: Returns true if the `isInitialized` property is gettable for the `disk` component class.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `diskisinitializedrateget`
Description: Returns a zero rate for the `isInitialized` property for the `disk` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `diskluminositiesstellar`
Description: Returns the default value for the `luminositiesStellar` property for the `disk` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `diskluminositiesstellarattributematch`
Description: Return a text list of component implementations in the `disk` class that have the desired attributes for the `luminositiesStellar` property
Code lines: 46
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `diskluminositiesstellarisgettable`
Description: Returns true if the `luminositiesStellar` property is gettable for the `disk` component class.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `diskluminositiesstellarrateget`
Description: Returns a zero rate for the `luminositiesStellar` property for the `disk` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `diskmassgas`
Description: Returns the default value for the `massGas` property for the `disk` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `diskmassgasattributematch`
Description: Return a text list of component implementations in the `disk` class that have the desired attributes for the `massGas` property
Code lines: 46
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `diskmassgasisgettable`
Description: Returns true if the `massGas` property is gettable for the `disk` component class.
Code lines: 8
Contained by: module `galacticus_nodes`

subroutine: `diskmassgasrate`
Description: Accept a rate set for the `massGas` property of the `disk` component class. Trigger an interrupt to create the component.
Code lines: 21
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

function: `diskmassgasrateget`
Description: Returns a zero rate for the `massGas` property for the `disk` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `diskmassstellar`
Description: Returns the default value for the `massStellar` property for the `disk` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `diskmassstellarattributematch`
Description: Return a text list of component implementations in the `disk` class that have the desired attributes for the `massStellar` property
Code lines: 46
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `diskmassstellarformed`
Description: Returns the default value for the `massStellarFormed` property for the `disk` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `diskmassstellarformedattributematch`
Description: Return a text list of component implementations in the `disk` class that have the desired attributes for the `massStellarFormed` property
Code lines: 28
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `diskmassstellarformedisgettable`
Description: Returns true if the `massStellarFormed` property is gettable for the `disk` component class.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `diskmassstellarformedrateget`
Description: Returns a zero rate for the `massStellarFormed` property for the `disk` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `diskmassstellarisgettable`
Description: Returns true if the `massStellar` property is gettable for the `disk` component class.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `diskmassstellarrateget`
Description: Returns a zero rate for the `massStellar` property for the `disk` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

subroutine: `diskknulloutputcount`
Description: Increment the count of properties to output for a `null` implementation of the `disk` component.
Code lines: 22
Contained by: module `galacticus_nodes`

subroutine: `diskknulloutputnames`
Description: Return the names of properties to output for a `null` implementation of the `disk` component.
Code lines: 39
Contained by: module `galacticus_nodes`

subroutine: `diskoutput`
Description: Populate output buffers with properties to output for a `disk` component.
Code lines: 74
Contained by: module `galacticus_nodes`
Modules used: `multi_counters`

subroutine: `diskoutputcount`
Description: Increment the count of properties to output for a generic `disk` component.
Code lines: 15
Contained by: module `galacticus_nodes`

subroutine: `diskoutputnames`
Description: Establish the names of properties to output for a generic `disk` component.
Code lines: 20
Contained by: module `galacticus_nodes`

subroutine: `diskpostoutput`

Description: Perform post-output processing of a `disk` component.
Code lines: 14
Contained by: module galacticus_nodes

function: `diskradius`
Description: Returns the default value for the `radius` property for the `disk` component class.
Code lines: 13
Contained by: module galacticus_nodes

function: `diskradiusattributematch`
Description: Return a text list of component implementations in the `disk` class that have the desired attributes for the `radius` property
Code lines: 41
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: `diskradiusisgettable`
Description: Returns true if the `radius` property is gettable for the `disk` component class.
Code lines: 8
Contained by: module galacticus_nodes

function: `diskradiusrateget`
Description: Returns a zero rate for the `radius` property for the `disk` component class.
Code lines: 13
Contained by: module galacticus_nodes

function: `diskstandardabundancesgascount`
Description: Return a count of the number of scalar properties in the `abundancesGas` property of an `standard` implementation of the `disk` component class.
Code lines: 12
Contained by: module galacticus_nodes

function: `diskstandardabundancesgasget`
Description: Get the `abundancesGas` property of an `standard` implementation of the `disk` component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: `diskstandardabundancesgasjcbnzr`
Description: Indicate that the `abundancesGas` property of an `standard` implementation of the `disk` component class is inactive for differential equation solving.
Code lines: 14
Contained by: module galacticus_nodes

subroutine: `diskstandardabundancesgasrate`
Description: Accumulate to the rate of change of the `abundancesGas` property of an `standard` implementation of the `disk` component class.
Code lines: 38
Contained by: module galacticus_nodes

subroutine: `diskstandardabundancesgasrategeneric`

Description: Set the rate of the `abundancesGas` property of the `abundancesGas` property of an `standard` implementation of the `disk` component class via a generic `nodeComponent`.
Code lines: 27
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

function: `diskstandardabundancesgasrateget`
Description: Get the rate of change of the `abundancesGas` property of an `standard` implementation of the `disk` component class.
Code lines: 26
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `diskstandardabundancesgasscale`
Description: Set the absolute scale of the `abundancesGas` property of an `standard` implementation of the `disk` component class.
Code lines: 15
Contained by: module `galacticus_nodes`

subroutine: `diskstandardabundancesgasset`
Description: Set the `abundancesGas` property of an `standard` implementation of the `disk` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `diskstandardabundancesstellarcount`
Description: Return a count of the number of scalar properties in the `abundancesStellar` property of an `standard` implementation of the `disk` component class.
Code lines: 12
Contained by: module `galacticus_nodes`

function: `diskstandardabundancesstellarget`
Description: Get the `abundancesStellar` property of an `standard` implementation of the `disk` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `diskstandardabundancesstellarjcbnzr`
Description: Indicate that the `abundancesStellar` property of an `standard` implementation of the `disk` component class is inactive for differential equation solving.
Code lines: 14
Contained by: module `galacticus_nodes`

subroutine: `diskstandardabundancesstellarrate`
Description: Accumulate to the rate of change of the `abundancesStellar` property of an `standard` implementation of the `disk` component class.
Code lines: 38
Contained by: module `galacticus_nodes`

function: `diskstandardabundancesstellarrateget`

Description: Get the rate of change of the `abundancesStellar` property of an `standard` implementation of the `disk` component class.
Code lines: 26
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: `diskstandardabundancesstellarscale`
Description: Set the absolute scale of the `abundancesStellar` property of an `standard` implementation of the `disk` component class.
Code lines: 15
Contained by: module galacticus_nodes

subroutine: `diskstandardabundancesstellarset`
Description: Set the `abundancesStellar` property of an `standard` implementation of the `disk` component class.
Code lines: 10
Contained by: module galacticus_nodes

function: `diskstandardangularmomentumcount`
Description: Return a count of the number of scalar properties in the `angularMomentum` property of an `standard` implementation of the `disk` component class.
Code lines: 12
Contained by: module galacticus_nodes

function: `diskstandardangularmomentumget`
Description: Get the `angularMomentum` property of an `standard` implementation of the `disk` component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: `diskstandardangularmomentumjcbnzr`
Description: Indicate that the `angularMomentum` property of an `standard` implementation of the `disk` component class is inactive for differential equation solving.
Code lines: 12
Contained by: module galacticus_nodes

subroutine: `diskstandardangularmomentumrate`
Description: Accumulate to the rate of change of the `angularMomentum` property of an `standard` implementation of the `disk` component class.
Code lines: 31
Contained by: module galacticus_nodes

subroutine: `diskstandardangularmomentumrategeneric`
Description: Set the rate of the `angularMomentum` property of the `angularMomentum` property of an `standard` implementation of the `disk` component class via a generic `nodeComponent`.
Code lines: 27
Contained by: module galacticus_nodes
Modules used: galacticus_error

function: `diskstandardangularmomentumrateget`

Description: Get the rate of change of the `angularMomentum` property of an `standard` implementation of the `disk` component class.
Code lines: 23
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `diskstandardangularmomentumscale`
Description: Set the absolute scale of the `angularMomentum` property of an `standard` implementation of the `disk` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

subroutine: `diskstandardangularmomentumset`
Description: Set the `angularMomentum` property of an `standard` implementation of the `disk` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `diskstandardfractionmassretainedcount`
Description: Return a count of the number of scalar properties in the `fractionMassRetained` property of an `standard` implementation of the `disk` component class.
Code lines: 12
Contained by: module `galacticus_nodes`

function: `diskstandardfractionmassretainedget`
Description: Get the `fractionMassRetained` property of an `standard` implementation of the `disk` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `diskstandardfractionmassretainedjcbnzr`
Description: Indicate that the `fractionMassRetained` property of an `standard` implementation of the `disk` component class is inactive for differential equation solving.
Code lines: 12
Contained by: module `galacticus_nodes`

subroutine: `diskstandardfractionmassretainedrate`
Description: Accumulate to the rate of change of the `fractionMassRetained` property of an `standard` implementation of the `disk` component class.
Code lines: 31
Contained by: module `galacticus_nodes`

function: `diskstandardfractionmassretainedrateget`
Description: Get the rate of change of the `fractionMassRetained` property of an `standard` implementation of the `disk` component class.
Code lines: 23
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `diskstandardfractionmassretainedscale`

Description: Set the absolute scale of the `fractionMassRetained` property of an `standard` implementation of the `disk` component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: `diskstandardfractionmassretainedset`
Description: Set the `fractionMassRetained` property of an `standard` implementation of the `disk` component class.
Code lines: 10
Contained by: module galacticus_nodes

function: `diskstandardisinitializedget`
Description: Get the `isInitialized` property of an `standard` implementation of the `disk` component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: `diskstandardisinitializedset`
Description: Set the `isInitialized` property of an `standard` implementation of the `disk` component class.
Code lines: 10
Contained by: module galacticus_nodes

function: `diskstandardluminositiesstellarcount`
Description: Return a count of the number of scalar properties in the `luminositiesStellar` property of an `standard` implementation of the `disk` component class.
Code lines: 12
Contained by: module galacticus_nodes

function: `diskstandardluminositiesstellarget`
Description: Get the `luminositiesStellar` property of an `standard` implementation of the `disk` component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: `diskstandardluminositiesstellarjcbnzr`
Description: Indicate that the `luminositiesStellar` property of an `standard` implementation of the `disk` component class is inactive for differential equation solving.
Code lines: 14
Contained by: module galacticus_nodes

subroutine: `diskstandardluminositiesstellarrate`
Description: Accumulate to the rate of change of the `luminositiesStellar` property of an `standard` implementation of the `disk` component class.
Code lines: 38
Contained by: module galacticus_nodes

function: `diskstandardluminositiesstellarrateget`
Description: Get the rate of change of the `luminositiesStellar` property of an `standard` implementation of the `disk` component class.
Code lines: 26
Contained by: module galacticus_nodes

Modules used: galacticus_error

subroutine: diskstandardluminositiesstellarscale
Description: Set the absolute scale of the luminositiesStellar property of an standard implementation of the disk component class.
Code lines: 15
Contained by: module galacticus_nodes

subroutine: diskstandardluminositiesstellarset
Description: Set the luminositiesStellar property of an standard implementation of the disk component class.
Code lines: 10
Contained by: module galacticus_nodes

function: diskstandardmassgascount
Description: Return a count of the number of scalar properties in the massGas property of an standard implementation of the disk component class.
Code lines: 12
Contained by: module galacticus_nodes

function: diskstandardmassgasget
Description: Get the massGas property of an standard implementation of the disk component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: diskstandardmassgasjcbnzr
Description: Indicate that the massGas property of an standard implementation of the disk component class is inactive for differential equation solving.
Code lines: 12
Contained by: module galacticus_nodes

subroutine: diskstandardmassgasrate
Description: Accumulate to the rate of change of the massGas property of an standard implementation of the disk component class.
Code lines: 31
Contained by: module galacticus_nodes

subroutine: diskstandardmassgasrategeneric
Description: Set the rate of the massGas property of the massGas property of an standard implementation of the disk component class via a generic nodeComponent.
Code lines: 27
Contained by: module galacticus_nodes
Modules used: galacticus_error

function: diskstandardmassgasrateget
Description: Get the rate of change of the massGas property of an standard implementation of the disk component class.
Code lines: 23
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: `diskstandardmassgasscale`
Description: Set the absolute scale of the `massGas` property of an `standard` implementation of the `disk` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

subroutine: `diskstandardmassgasset`
Description: Set the `massGas` property of an `standard` implementation of the `disk` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `diskstandardmassstellarcount`
Description: Return a count of the number of scalar properties in the `massStellar` property of an `standard` implementation of the `disk` component class.
Code lines: 12
Contained by: module `galacticus_nodes`

function: `diskstandardmassstellarformedcount`
Description: Return a count of the number of scalar properties in the `massStellarFormed` property of an `standard` implementation of the `disk` component class.
Code lines: 12
Contained by: module `galacticus_nodes`

function: `diskstandardmassstellarformedget`
Description: Get the `massStellarFormed` property of an `standard` implementation of the `disk` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `diskstandardmassstellarformedjcbnzr`
Description: Indicate that the `massStellarFormed` property of an `standard` implementation of the `disk` component class is inactive for differential equation solving.
Code lines: 12
Contained by: module `galacticus_nodes`

subroutine: `diskstandardmassstellarformedrate`
Description: Accumulate to the rate of change of the `massStellarFormed` property of an `standard` implementation of the `disk` component class.
Code lines: 31
Contained by: module `galacticus_nodes`

function: `diskstandardmassstellarformedrateget`
Description: Get the rate of change of the `massStellarFormed` property of an `standard` implementation of the `disk` component class.
Code lines: 23
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `diskstandardmassstellarformedscale`

Description: Set the absolute scale of the `massStellarFormed` property of an `standard` implementation of the `disk` component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: `diskstandardmassstellarformedset`
Description: Set the `massStellarFormed` property of an `standard` implementation of the `disk` component class.
Code lines: 10
Contained by: module galacticus_nodes

function: `diskstandardmassstellarget`
Description: Get the `massStellar` property of an `standard` implementation of the `disk` component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: `diskstandardmassstellarjcbnzr`
Description: Indicate that the `massStellar` property of an `standard` implementation of the `disk` component class is inactive for differential equation solving.
Code lines: 12
Contained by: module galacticus_nodes

subroutine: `diskstandardmassstellarrate`
Description: Accumulate to the rate of change of the `massStellar` property of an `standard` implementation of the `disk` component class.
Code lines: 31
Contained by: module galacticus_nodes

function: `diskstandardmassstellarrateget`
Description: Get the rate of change of the `massStellar` property of an `standard` implementation of the `disk` component class.
Code lines: 23
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: `diskstandardmassstellarscale`
Description: Set the absolute scale of the `massStellar` property of an `standard` implementation of the `disk` component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: `diskstandardmassstellarset`
Description: Set the `massStellar` property of an `standard` implementation of the `disk` component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: `diskstandardoutputcount`
Description: Increment the count of properties to output for a `standard` implementation of the `disk` component.
Code lines: 24
Contained by: module galacticus_nodes

subroutine: `diskstandardoutputnames`
Description: Return the names of properties to output for a `standard` implementation of the `disk` component.
Code lines: 71
Contained by: module `galacticus_nodes`

subroutine: `diskstandardpostoutput`
Description: Perform post-output processing for a `standard` implementation of the `disk` component.
Code lines: 12
Contained by: module `galacticus_nodes`

function: `diskstandardradiusget`
Description: Get the `radius` property of an `standard` implementation of the `disk` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `diskstandardradiusset`
Description: Set the `radius` property of an `standard` implementation of the `disk` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `diskstandardstarformationhistorycount`
Description: Return a count of the number of scalar properties in the `starFormationHistory` property of an `standard` implementation of the `disk` component class.
Code lines: 12
Contained by: module `galacticus_nodes`

function: `diskstandardstarformationhistoryget`
Description: Get the `starFormationHistory` property of an `standard` implementation of the `disk` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `diskstandardstarformationhistoryjcbnzr`
Description: Indicate that the `starFormationHistory` property of an `standard` implementation of the `disk` component class is inactive for differential equation solving.
Code lines: 14
Contained by: module `galacticus_nodes`

subroutine: `diskstandardstarformationhistoryrate`
Description: Accumulate to the rate of change of the `starFormationHistory` property of an `standard` implementation of the `disk` component class.
Code lines: 38
Contained by: module `galacticus_nodes`

function: `diskstandardstarformationhistoryrateget`
Description: Get the rate of change of the `starFormationHistory` property of an `standard` implementation of the `disk` component class.
Code lines: 26

Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: diskstandardstarformationhistoryscale
Description: Set the absolute scale of the starFormationHistory property of an standard implementation of the disk component class.
Code lines: 15
Contained by: module galacticus_nodes

subroutine: diskstandardstarformationhistoryset
Description: Set the starFormationHistory property of an standard implementation of the disk component class.
Code lines: 10
Contained by: module galacticus_nodes

function: diskstandardstarformationrateget
Description: Get the value of the starFormationRate property of the standard implementation of the disk component using a deferred function.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: diskstandardstarformationrategetfunction
Description: Set the function to be used for the get method of the starFormationRate property of the DiskStandard component.
Code lines: 10
Contained by: module galacticus_nodes

function: diskstandardstarformationrategetisattached
Description: Return true if the deferred function used to get the starFormationRate property of the DiskStandard component class has been attached.
Code lines: 8
Contained by: module galacticus_nodes

function: diskstandardstellarpropertieshistorycount
Description: Return a count of the number of scalar properties in the stellarPropertiesHistory property of an standard implementation of the disk component class.
Code lines: 12
Contained by: module galacticus_nodes

function: diskstandardstellarpropertieshistoryget
Description: Get the stellarPropertiesHistory property of an standard implementation of the disk component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: diskstandardstellarpropertieshistoryjcbnzr
Description: Indicate that the stellarPropertiesHistory property of an standard implementation of the disk component class is inactive for differential equation solving.
Code lines: 14
Contained by: module galacticus_nodes

subroutine: `diskstandardstellarpropertieshistoryrate`
Description: Accumulate to the rate of change of the `stellarPropertiesHistory` property of an `standard` implementation of the `disk` component class.
Code lines: 38
Contained by: module galacticus_nodes

function: `diskstandardstellarpropertieshistoryrateget`
Description: Get the rate of change of the `stellarPropertiesHistory` property of an `standard` implementation of the `disk` component class.
Code lines: 26
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: `diskstandardstellarpropertieshistoryscale`
Description: Set the absolute scale of the `stellarPropertiesHistory` property of an `standard` implementation of the `disk` component class.
Code lines: 15
Contained by: module galacticus_nodes

subroutine: `diskstandardstellarpropertieshistoryset`
Description: Set the `stellarPropertiesHistory` property of an `standard` implementation of the `disk` component class.
Code lines: 10
Contained by: module galacticus_nodes

function: `diskstandardvelocityget`
Description: Get the `velocity` property of an `standard` implementation of the `disk` component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: `diskstandardvelocityset`
Description: Set the `velocity` property of an `standard` implementation of the `disk` component class.
Code lines: 10
Contained by: module galacticus_nodes

function: `diskstarformationhistory`
Description: Returns the default value for the `starFormationHistory` property for the `disk` component class.
Code lines: 13
Contained by: module galacticus_nodes

function: `diskstarformationhistoryattributematch`
Description: Return a text list of component implementations in the `disk` class that have the desired attributes for the `starFormationHistory` property
Code lines: 28
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: `diskstarformationhistoryisgettable`

Description: Returns true if the `starFormationHistory` property is gettable for the `disk` component class.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `diskstarformationhistoryrateget`
Description: Returns a zero rate for the `starFormationHistory` property for the `disk` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `diskstarformationrate`
Description: Returns the default value for the `starFormationRate` property for the `disk` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `diskstarformationrateattributematch`
Description: Return a text list of component implementations in the `disk` class that have the desired attributes for the `starFormationRate` property
Code lines: 52
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `diskstarformationrateisgettable`
Description: Returns true if the `starFormationRate` property is gettable for the `disk` component class.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `diskstarformationraterateget`
Description: Returns a zero rate for the `starFormationRate` property for the `disk` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `diskstellarpropertieshistory`
Description: Returns the default value for the `stellarPropertiesHistory` property for the `disk` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `diskstellarpropertieshistoryattributematch`
Description: Return a text list of component implementations in the `disk` class that have the desired attributes for the `stellarPropertiesHistory` property
Code lines: 46
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `diskstellarpropertieshistoryisgettable`
Description: Returns true if the `stellarPropertiesHistory` property is gettable for the `disk` component class.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `diskstellarpropertieshistoryrateget`
Description: Returns a zero rate for the `stellarPropertiesHistory` property for the `disk` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `diskvelocity`
Description: Returns the default value for the `velocity` property for the `disk` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `diskvelocityattributematch`
Description: Return a text list of component implementations in the `disk` class that have the desired attributes for the `velocity` property
Code lines: 41
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `diskvelocityisgettable`
Description: Returns true if the `velocity` property is gettable for the `disk` component class.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `diskvelocityrateget`
Description: Returns a zero rate for the `velocity` property for the `disk` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `diskverysimpleabundancesgascount`
Description: Return a count of the number of scalar properties in the `abundancesGas` property of an `verySimple` implementation of the `disk` component class.
Code lines: 12
Contained by: module `galacticus_nodes`

function: `diskverysimpleabundancesgasget`
Description: Get the `abundancesGas` property of an `verySimple` implementation of the `disk` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `diskverysimpleabundancesgasjcbnzr`
Description: Indicate that the `abundancesGas` property of an `verySimple` implementation of the `disk` component class is inactive for differential equation solving.
Code lines: 14
Contained by: module `galacticus_nodes`

subroutine: `diskverysimpleabundancesgasrate`
Description: Accumulate to the rate of change of the `abundancesGas` property of an `verySimple` implementation of the `disk` component class.
Code lines: 38

Contained by: module galacticus_nodes

subroutine: diskverysimpleabundancesgasrategeneric
Description: Set the rate of the abundancesGas property of the abundancesGas property of an verySimple implementation of the disk component class via a generic nodeComponent.
Code lines: 27
Contained by: module galacticus_nodes
Modules used: galacticus_error

function: diskverysimpleabundancesgasrateget
Description: Get the rate of change of the abundancesGas property of an verySimple implementation of the disk component class.
Code lines: 26
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: diskverysimpleabundancesgasscale
Description: Set the absolute scale of the abundancesGas property of an verySimple implementation of the disk component class.
Code lines: 15
Contained by: module galacticus_nodes

subroutine: diskverysimpleabundancesgasset
Description: Set the abundancesGas property of an verySimple implementation of the disk component class.
Code lines: 10
Contained by: module galacticus_nodes

function: diskverysimpleabundancesstellarcount
Description: Return a count of the number of scalar properties in the abundancesStellar property of an verySimple implementation of the disk component class.
Code lines: 12
Contained by: module galacticus_nodes

function: diskverysimpleabundancesstellarget
Description: Get the abundancesStellar property of an verySimple implementation of the disk component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: diskverysimpleabundancesstellarjcbnzr
Description: Indicate that the abundancesStellar property of an verySimple implementation of the disk component class is inactive for differential equation solving.
Code lines: 14
Contained by: module galacticus_nodes

subroutine: diskverysimpleabundancesstellarrate
Description: Accumulate to the rate of change of the abundancesStellar property of an verySimple implementation of the disk component class.
Code lines: 38
Contained by: module galacticus_nodes

function: `diskverysimpleabundancesstellarrateget`
Description: Get the rate of change of the `abundancesStellar` property of an `verySimple` implementation of the `disk` component class.
Code lines: 26
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `diskverysimpleabundancesstellarscale`
Description: Set the absolute scale of the `abundancesStellar` property of an `verySimple` implementation of the `disk` component class.
Code lines: 15
Contained by: module `galacticus_nodes`

subroutine: `diskverysimpleabundancesstellarset`
Description: Set the `abundancesStellar` property of an `verySimple` implementation of the `disk` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `diskverysimpleisinitializedget`
Description: Get the `isInitialized` property of an `verySimple` implementation of the `disk` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `diskverysimpleisinitializedset`
Description: Set the `isInitialized` property of an `verySimple` implementation of the `disk` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `diskverysimpleluminositiesstellarcount`
Description: Return a count of the number of scalar properties in the `luminositiesStellar` property of an `verySimple` implementation of the `disk` component class.
Code lines: 12
Contained by: module `galacticus_nodes`

function: `diskverysimpleluminositiesstellarget`
Description: Get the `luminositiesStellar` property of an `verySimple` implementation of the `disk` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `diskverysimpleluminositiesstellarjcbnzr`
Description: Indicate that the `luminositiesStellar` property of an `verySimple` implementation of the `disk` component class is inactive for differential equation solving.
Code lines: 14
Contained by: module `galacticus_nodes`

subroutine: `diskverysimpleluminositiesstellarrate`

Description: Accumulate to the rate of change of the `luminositiesStellar` property of an `verySimple` implementation of the `disk` component class.
Code lines: 38
Contained by: module `galacticus_nodes`

function: `diskverysimpleluminositiesstellarrateget`
Description: Get the rate of change of the `luminositiesStellar` property of an `verySimple` implementation of the `disk` component class.
Code lines: 26
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `diskverysimpleluminositiesstellarscale`
Description: Set the absolute scale of the `luminositiesStellar` property of an `verySimple` implementation of the `disk` component class.
Code lines: 15
Contained by: module `galacticus_nodes`

subroutine: `diskverysimpleluminositiesstellarset`
Description: Set the `luminositiesStellar` property of an `verySimple` implementation of the `disk` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `diskverysimplemassgascount`
Description: Return a count of the number of scalar properties in the `massGas` property of an `verySimple` implementation of the `disk` component class.
Code lines: 12
Contained by: module `galacticus_nodes`

function: `diskverysimplemassgasget`
Description: Get the `massGas` property of an `verySimple` implementation of the `disk` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `diskverysimplemassgasjcbnzr`
Description: Indicate that the `massGas` property of an `verySimple` implementation of the `disk` component class is inactive for differential equation solving.
Code lines: 12
Contained by: module `galacticus_nodes`

subroutine: `diskverysimplemassgasrate`
Description: Accumulate to the rate of change of the `massGas` property of an `verySimple` implementation of the `disk` component class.
Code lines: 31
Contained by: module `galacticus_nodes`

subroutine: `diskverysimplemassgasrategeneric`
Description: Set the rate of the `massGas` property of the `massGas` property of an `verySimple` implementation of the `disk` component class via a generic `nodeComponent`.
Code lines: 27

Contained by: module galacticus_nodes
Modules used: galacticus_error

function: diskverysimplemassgasrateget
Description: Get the rate of change of the massGas property of an verySimple implementation of the disk component class.
Code lines: 23
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: diskverysimplemassgasscale
Description: Set the absolute scale of the massGas property of an verySimple implementation of the disk component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: diskverysimplemassgasset
Description: Set the massGas property of an verySimple implementation of the disk component class.
Code lines: 10
Contained by: module galacticus_nodes

function: diskverysimplemassstellarcount
Description: Return a count of the number of scalar properties in the massStellar property of an verySimple implementation of the disk component class.
Code lines: 12
Contained by: module galacticus_nodes

function: diskverysimplemassstellarget
Description: Get the massStellar property of an verySimple implementation of the disk component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: diskverysimplemassstellarjcbnzr
Description: Indicate that the massStellar property of an verySimple implementation of the disk component class is inactive for differential equation solving.
Code lines: 12
Contained by: module galacticus_nodes

subroutine: diskverysimplemassstellarrate
Description: Accumulate to the rate of change of the massStellar property of an verySimple implementation of the disk component class.
Code lines: 31
Contained by: module galacticus_nodes

function: diskverysimplemassstellarrateget
Description: Get the rate of change of the massStellar property of an verySimple implementation of the disk component class.
Code lines: 23
Contained by: module galacticus_nodes

Modules used: galacticus_error

subroutine: diskverysimplemassstellarscale
Description: Set the absolute scale of the massStellar property of an verySimple implementation of the disk component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: diskverysimplemassstellarset
Description: Set the massStellar property of an verySimple implementation of the disk component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: diskverysimpleoutputcount
Description: Increment the count of properties to output for a verySimple implementation of the disk component.
Code lines: 24
Contained by: module galacticus_nodes

subroutine: diskverysimpleoutputnames
Description: Return the names of properties to output for a verySimple implementation of the disk component.
Code lines: 53
Contained by: module galacticus_nodes

subroutine: diskverysimplepostoutput
Description: Perform post-output processing for a verySimple implementation of the disk component.
Code lines: 12
Contained by: module galacticus_nodes

subroutine: diskverysimplesizeoutputcount
Description: Increment the count of properties to output for a verySimpleSize implementation of the disk component.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: diskverysimplesizeoutputnames
Description: Return the names of properties to output for a verySimpleSize implementation of the disk component.
Code lines: 29
Contained by: module galacticus_nodes

subroutine: diskverysimplesizepostoutput
Description: Perform post-output processing for a verySimpleSize implementation of the disk component.
Code lines: 10
Contained by: module galacticus_nodes

function: diskverysimplesizeradiusget

Description: Get the `radius` property of an `verySimpleSize` implementation of the `disk` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `diskverysimplesizeradiusset`
Description: Set the `radius` property of an `verySimpleSize` implementation of the `disk` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `diskverysimplesizevelocityget`
Description: Get the `velocity` property of an `verySimpleSize` implementation of the `disk` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `diskverysimplesizevelocityset`
Description: Set the `velocity` property of an `verySimpleSize` implementation of the `disk` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `diskverysimplestarformationrateget`
Description: Get the value of the `starFormationRate` property of the `verySimple` implementation of the `disk` component using a deferred function.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `diskverysimplestarformationrategetfunction`
Description: Set the function to be used for the `get` method of the `starFormationRate` property of the `DiskVerySimple` component.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `diskverysimplestarformationrategetisattached`
Description: Return true if the deferred function used to get the `starFormationRate` property of the `DiskVerySimple` component class has been attached.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `diskverysimplestellarpropertieshistorycount`
Description: Return a count of the number of scalar properties in the `stellarPropertiesHistory` property of an `verySimple` implementation of the `disk` component class.
Code lines: 12
Contained by: module `galacticus_nodes`

function: `diskverysimplestellarpropertieshistoryget`
Description: Get the `stellarPropertiesHistory` property of an `verySimple` implementation of the `disk` component class.
Code lines: 10

Contained by: module galacticus_nodes

subroutine: diskverysimplestellarpropertieshistoryjcbnzr
Description: Indicate that the stellarPropertiesHistory property of an verySimple implementation of the disk component class is inactive for differential equation solving.
Code lines: 14
Contained by: module galacticus_nodes

subroutine: diskverysimplestellarpropertieshistoryrate
Description: Accumulate to the rate of change of the stellarPropertiesHistory property of an verySimple implementation of the disk component class.
Code lines: 38
Contained by: module galacticus_nodes

function: diskverysimplestellarpropertieshistoryrateget
Description: Get the rate of change of the stellarPropertiesHistory property of an verySimple implementation of the disk component class.
Code lines: 26
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: diskverysimplestellarpropertieshistoryscale
Description: Set the absolute scale of the stellarPropertiesHistory property of an verySimple implementation of the disk component class.
Code lines: 15
Contained by: module galacticus_nodes

subroutine: diskverysimplestellarpropertieshistoryset
Description: Set the stellarPropertiesHistory property of an verySimple implementation of the disk component class.
Code lines: 10
Contained by: module galacticus_nodes

function: dynamicsstatisticsadiabaticratio
Description: Returns the default value for the adiabaticRatio property for the dynamicsStatistics component class.
Code lines: 13
Contained by: module galacticus_nodes

function: dynamicsstatisticsadiabaticratioattributematch
Description: Return a text list of component implementations in the dynamicsStatistics class that have the desired attributes for the adiabaticRatio property
Code lines: 30
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: dynamicsstatisticsadiabaticratioisgettable
Description: Returns true if the adiabaticRatio property is gettable for the dynamicsStatistics component class.
Code lines: 8
Contained by: module galacticus_nodes

function: `dynamicsstatisticsadiabaticratiorateget`
Description: Returns a zero rate for the `adiabaticRatio` property for the `dynamicsStatistics` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `dynamicsstatisticsbarinstabilitytimescale`
Description: Returns the default value for the `barInstabilityTimescale` property for the `dynamicsStatistics` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `dynamicsstatisticsbarinstabilitytimescaleattributematch`
Description: Return a text list of component implementations in the `dynamicsStatistics` class that have the desired attributes for the `barInstabilityTimescale` property
Code lines: 30
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `dynamicsstatisticsbarinstabilitytimescaleisgettable`
Description: Returns true if the `barInstabilityTimescale` property is gettable for the `dynamicsStatistics` component class.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `dynamicsstatisticsbarinstabilitytimescalerateget`
Description: Returns a zero rate for the `barInstabilityTimescale` property for the `dynamicsStatistics` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `dynamicsstatisticsbarsadiabaticratioget`
Description: Get the `adiabaticRatio` property of an `bars` implementation of the `dynamicsStatistics` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `dynamicsstatisticsbarsadiabaticratioset`
Description: Set the `adiabaticRatio` property of an `bars` implementation of the `dynamicsStatistics` component class.
Code lines: 20
Contained by: module `galacticus_nodes`
Modules used: `memory_management`

function: `dynamicsstatisticsbarsbarinstabilitytimescaleget`
Description: Get the `barInstabilityTimescale` property of an `bars` implementation of the `dynamicsStatistics` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `dynamicsstatisticsbarsbarinstabilitytimescaleset`
Description: Set the `barInstabilityTimescale` property of an `bars` implementation of the `dynamicsStatistics` component class.
Code lines: 20
Contained by: module `galacticus_nodes`
Modules used: `memory_management`

subroutine: `dynamicsstatisticsbarsoutputcount`
Description: Increment the count of properties to output for a `bars` implementation of the `dynamicsStatistics` component.
Code lines: 22
Contained by: module `galacticus_nodes`

subroutine: `dynamicsstatisticsbarsoutputnames`
Description: Return the names of properties to output for a `bars` implementation of the `dynamicsStatistics` component.
Code lines: 39
Contained by: module `galacticus_nodes`

function: `dynamicsstatisticsbarstimeget`
Description: Get the `time` property of an `bars` implementation of the `dynamicsStatistics` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `dynamicsstatisticsbarstimeset`
Description: Set the `time` property of an `bars` implementation of the `dynamicsStatistics` component class.
Code lines: 20
Contained by: module `galacticus_nodes`
Modules used: `memory_management`

subroutine: `dynamicsstatisticsnulloutputcount`
Description: Increment the count of properties to output for a `null` implementation of the `dynamicsStatistics` component.
Code lines: 22
Contained by: module `galacticus_nodes`

subroutine: `dynamicsstatisticsnulloutputnames`
Description: Return the names of properties to output for a `null` implementation of the `dynamicsStatistics` component.
Code lines: 39
Contained by: module `galacticus_nodes`

subroutine: `dynamicsstatisticsoutput`
Description: Populate output buffers with properties to output for a `dynamicsStatistics` component.
Code lines: 38
Contained by: module `galacticus_nodes`
Modules used: `multi_counters`

subroutine: `dynamicsstatisticsoutputcount`
Description: Increment the count of properties to output for a generic `dynamicsStatistics` component.
Code lines: 15
Contained by: module `galacticus_nodes`

subroutine: `dynamicsstatisticsoutputnames`
Description: Establish the names of properties to output for a generic `dynamicsStatistics` component.
Code lines: 20
Contained by: module `galacticus_nodes`

subroutine: `dynamicsstatisticspostoutput`
Description: Perform post-output processing of a `dynamicsStatistics` component.
Code lines: 14
Contained by: module `galacticus_nodes`

function: `dynamicsstatisticstime`
Description: Returns the default value for the `time` property for the `dynamicsStatistics` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `dynamicsstatisticstimeattributematch`
Description: Return a text list of component implementations in the `dynamicsStatistics` class that have the desired attributes for the `time` property
Code lines: 30
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `dynamicsstatisticstimeisgettable`
Description: Returns true if the `time` property is gettable for the `dynamicsStatistics` component class.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `dynamicsstatisticstimerateget`
Description: Returns a zero rate for the `time` property for the `dynamicsStatistics` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `formationtimecole2000formationtime`
Description: Return the `formationTime` property of the `formationTime` component class.
Code lines: 25
Contained by: module `galacticus_nodes`

subroutine: `formationtimecole2000outputcount`
Description: Increment the count of properties to output for a `Cole2000` implementation of the `formationTime` component.
Code lines: 22
Contained by: module `galacticus_nodes`

subroutine: `formationtimecole2000outputnames`
Description: Return the names of properties to output for a `Cole2000` implementation of the `formationTime` component.
Code lines: 39
Contained by: module galacticus_nodes

function: `formationtimeformationtime`
Description: Returns the default value for the `formationTime` property for the `formationTime` component class.
Code lines: 13
Contained by: module galacticus_nodes

function: `formationtimeformationtimeattributematch`
Description: Return a text list of component implementations in the `formationTime` class that have the desired attributes for the `formationTime` property
Code lines: 41
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: `formationtimeformationtimeisgettable`
Description: Returns true if the `formationTime` property is gettable for the `formationTime` component class.
Code lines: 8
Contained by: module galacticus_nodes

function: `formationtimeformationtimerateget`
Description: Returns a zero rate for the `formationTime` property for the `formationTime` component class.
Code lines: 13
Contained by: module galacticus_nodes

function: `formationtimemassfractionformationtimeget`
Description: Get the `formationTime` property of an `massFraction` implementation of the `formationTime` component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: `formationtimemassfractionformationtimeset`
Description: Set the `formationTime` property of an `massFraction` implementation of the `formationTime` component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: `formationtimemassfractionoutputcount`
Description: Increment the count of properties to output for a `massFraction` implementation of the `formationTime` component.
Code lines: 21
Contained by: module galacticus_nodes

subroutine: `formationtimemassfractionoutputnames`

Description: Return the names of properties to output for a `massFraction` implementation of the `formationTime` component.
Code lines: 37
Contained by: module galacticus_nodes

subroutine: `formationtimenulloutputcount`
Description: Increment the count of properties to output for a `null` implementation of the `formationTime` component.
Code lines: 22
Contained by: module galacticus_nodes

subroutine: `formationtimenulloutputnames`
Description: Return the names of properties to output for a `null` implementation of the `formationTime` component.
Code lines: 39
Contained by: module galacticus_nodes

subroutine: `formationtimeoutput`
Description: Populate output buffers with properties to output for a `formationTime` component.
Code lines: 34
Contained by: module galacticus_nodes
Modules used: multi_counters

subroutine: `formationtimeoutputcount`
Description: Increment the count of properties to output for a generic `formationTime` component.
Code lines: 15
Contained by: module galacticus_nodes

subroutine: `formationtimeoutputnames`
Description: Establish the names of properties to output for a generic `formationTime` component.
Code lines: 20
Contained by: module galacticus_nodes

subroutine: `formationtimepostoutput`
Description: Perform post-output processing of a `formationTime` component.
Code lines: 14
Contained by: module galacticus_nodes

subroutine: `galacticus_nodes_unique_id_set`
Description: Resets the global unique ID number.
Code lines: 7
Contained by: module galacticus_nodes

function: `hosthistoryhostmassmaximum`
Description: Returns the default value for the `hostMassMaximum` property for the `hostHistory` component class.
Code lines: 17
Contained by: module galacticus_nodes

function: `hosthistoryhostmassmaximumattributematch`

Description: Return a text list of component implementations in the `hostHistory` class that have the desired attributes for the `hostMassMaximum` property
Code lines: 30
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `hosthistoryhostmassmaximumisgettable`
Description: Returns true if the `hostMassMaximum` property is gettable for the `hostHistory` component class.
Code lines: 9
Contained by: module `galacticus_nodes`

function: `hosthistoryhostmassmaximumrateget`
Description: Returns a zero rate for the `hostMassMaximum` property for the `hostHistory` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

subroutine: `hosthistorynulloutputcount`
Description: Increment the count of properties to output for a `null` implementation of the `hostHistory` component.
Code lines: 22
Contained by: module `galacticus_nodes`

subroutine: `hosthistorynulloutputnames`
Description: Return the names of properties to output for a `null` implementation of the `hostHistory` component.
Code lines: 39
Contained by: module `galacticus_nodes`

subroutine: `hosthistoryoutput`
Description: Populate output buffers with properties to output for a `hostHistory` component.
Code lines: 34
Contained by: module `galacticus_nodes`
Modules used: `multi_counters`

subroutine: `hosthistoryoutputcount`
Description: Increment the count of properties to output for a generic `hostHistory` component.
Code lines: 15
Contained by: module `galacticus_nodes`

subroutine: `hosthistoryoutputnames`
Description: Establish the names of properties to output for a generic `hostHistory` component.
Code lines: 20
Contained by: module `galacticus_nodes`

subroutine: `hosthistorypostoutput`
Description: Perform post-output processing of a `hostHistory` component.
Code lines: 14
Contained by: module `galacticus_nodes`

function: `hosthistorystandardhostmassmaximumget`
Description: Get the `hostMassMaximum` property of an `standard` implementation of the `hostHistory` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `hosthistorystandardhostmassmaximumset`
Description: Set the `hostMassMaximum` property of an `standard` implementation of the `hostHistory` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `hosthistorystandardoutputcount`
Description: Increment the count of properties to output for a `standard` implementation of the `hostHistory` component.
Code lines: 21
Contained by: module `galacticus_nodes`

subroutine: `hosthistorystandardoutputnames`
Description: Return the names of properties to output for a `standard` implementation of the `hostHistory` component.
Code lines: 37
Contained by: module `galacticus_nodes`

function: `hothaloabundances`
Description: Returns the default value for the `abundances` property for the `hotHalo` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `hothaloabundancesattributematch`
Description: Return a text list of component implementations in the `hotHalo` class that have the desired attributes for the `abundances` property
Code lines: 64
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `hothaloabundancescold`
Description: Returns the default value for the `abundancesCold` property for the `hotHalo` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `hothaloabundancescoldattributematch`
Description: Return a text list of component implementations in the `hotHalo` class that have the desired attributes for the `abundancesCold` property
Code lines: 28
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `hothaloabundancescoldisgettable`

Description: Returns true if the `abundancesCold` property is gettable for the `hotHalo` component class.
Code lines: 8
Contained by: module `galacticus_nodes`

subroutine: `hothaloabundancescoldrate`
Description: Accept a rate set for the `abundancesCold` property of the `hotHalo` component class. Trigger an interrupt to create the component.
Code lines: 21
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

function: `hothaloabundancescoldrateget`
Description: Returns a zero rate for the `abundancesCold` property for the `hotHalo` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `hothaloabundancesisgettable`
Description: Returns true if the `abundances` property is gettable for the `hotHalo` component class.
Code lines: 8
Contained by: module `galacticus_nodes`

subroutine: `hothaloabundancesrate`
Description: Accept a rate set for the `abundances` property of the `hotHalo` component class. Trigger an interrupt to create the component.
Code lines: 21
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

function: `hothaloabundancesrateget`
Description: Returns a zero rate for the `abundances` property for the `hotHalo` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `hothaloangularmomentum`
Description: Returns the default value for the `angularMomentum` property for the `hotHalo` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `hothaloangularmomentumattributematch`
Description: Return a text list of component implementations in the `hotHalo` class that have the desired attributes for the `angularMomentum` property
Code lines: 46
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `hothaloangularmomentumcold`
Description: Returns the default value for the `angularMomentumCold` property for the `hotHalo` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `hothaloangularmomentumcoldattributematch`
Description: Return a text list of component implementations in the `hotHalo` class that have the desired attributes for the `angularMomentumCold` property
Code lines: 28
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `hothaloangularmomentumcoldisgettable`
Description: Returns true if the `angularMomentumCold` property is gettable for the `hotHalo` component class.
Code lines: 8
Contained by: module `galacticus_nodes`

subroutine: `hothaloangularmomentumcoldrate`
Description: Accept a rate set for the `angularMomentumCold` property of the `hotHalo` component class. Trigger an interrupt to create the component.
Code lines: 21
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

function: `hothaloangularmomentumcoldrateget`
Description: Returns a zero rate for the `angularMomentumCold` property for the `hotHalo` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `hothaloangularmomentumisgettable`
Description: Returns true if the `angularMomentum` property is gettable for the `hotHalo` component class.
Code lines: 8
Contained by: module `galacticus_nodes`

subroutine: `hothaloangularmomentumrate`
Description: Accept a rate set for the `angularMomentum` property of the `hotHalo` component class. Trigger an interrupt to create the component.
Code lines: 21
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

function: `hothaloangularmomentumrateget`
Description: Returns a zero rate for the `angularMomentum` property for the `hotHalo` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `hothalochemicals`
Description: Returns the default value for the `chemicals` property for the `hotHalo` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `hothalochemicalsattributematch`
Description: Return a text list of component implementations in the `hotHalo` class that have the desired attributes for the `chemicals` property
Code lines: 46
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `hothalochemicalsisgettable`
Description: Returns true if the `chemicals` property is gettable for the `hotHalo` component class.
Code lines: 8
Contained by: module `galacticus_nodes`

subroutine: `hothalochemicalsrate`
Description: Accept a rate set for the `chemicals` property of the `hotHalo` component class. Trigger an interrupt to create the component.
Code lines: 21
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

function: `hothalochemicalsrateget`
Description: Returns a zero rate for the `chemicals` property for the `hotHalo` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `hothalocoldmodeabundancescoldcount`
Description: Return a count of the number of scalar properties in the `abundancesCold` property of an `coldMode` implementation of the `hotHalo` component class.
Code lines: 12
Contained by: module `galacticus_nodes`

function: `hothalocoldmodeabundancescoldget`
Description: Get the `abundancesCold` property of an `coldMode` implementation of the `hotHalo` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `hothalocoldmodeabundancescoldjcbnzr`
Description: Indicate that the `abundancesCold` property of an `coldMode` implementation of the `hotHalo` component class is inactive for differential equation solving.
Code lines: 14
Contained by: module `galacticus_nodes`

subroutine: `hothalocoldmodeabundancescoldrate`
Description: Accumulate to the rate of change of the `abundancesCold` property of an `coldMode` implementation of the `hotHalo` component class.
Code lines: 38
Contained by: module `galacticus_nodes`

function: `hothalocoldmodeabundancescoldrateget`

Description: Get the rate of change of the `abundancesCold` property of an `coldMode` implementation of the `hotHalo` component class.
Code lines: 26
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: `hothalocoldmodeabundancescoldscale`
Description: Set the absolute scale of the `abundancesCold` property of an `coldMode` implementation of the `hotHalo` component class.
Code lines: 15
Contained by: module galacticus_nodes

subroutine: `hothalocoldmodeabundancescoldset`
Description: Set the `abundancesCold` property of an `coldMode` implementation of the `hotHalo` component class.
Code lines: 10
Contained by: module galacticus_nodes

function: `hothalocoldmodeangularmomentumcoldcount`
Description: Return a count of the number of scalar properties in the `angularMomentumCold` property of an `coldMode` implementation of the `hotHalo` component class.
Code lines: 12
Contained by: module galacticus_nodes

function: `hothalocoldmodeangularmomentumcoldget`
Description: Get the `angularMomentumCold` property of an `coldMode` implementation of the `hotHalo` component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: `hothalocoldmodeangularmomentumcoldjcbnzr`
Description: Indicate that the `angularMomentumCold` property of an `coldMode` implementation of the `hotHalo` component class is inactive for differential equation solving.
Code lines: 12
Contained by: module galacticus_nodes

subroutine: `hothalocoldmodeangularmomentumcoldrate`
Description: Accumulate to the rate of change of the `angularMomentumCold` property of an `coldMode` implementation of the `hotHalo` component class.
Code lines: 31
Contained by: module galacticus_nodes

function: `hothalocoldmodeangularmomentumcoldrateget`
Description: Get the rate of change of the `angularMomentumCold` property of an `coldMode` implementation of the `hotHalo` component class.
Code lines: 23
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: `hothalocoldmodeangularmomentumcoldscale`

Description: Set the absolute scale of the `angularMomentumCold` property of an `coldMode` implementation of the `hotHalo` component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: `hothalocoldmodeangularmomentumcoldset`
Description: Set the `angularMomentumCold` property of an `coldMode` implementation of the `hotHalo` component class.
Code lines: 10
Contained by: module galacticus_nodes

function: `hothalocoldmodemasscoldcount`
Description: Return a count of the number of scalar properties in the `massCold` property of an `coldMode` implementation of the `hotHalo` component class.
Code lines: 12
Contained by: module galacticus_nodes

function: `hothalocoldmodemasscoldget`
Description: Get the `massCold` property of an `coldMode` implementation of the `hotHalo` component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: `hothalocoldmodemasscoldjcbnzr`
Description: Indicate that the `massCold` property of an `coldMode` implementation of the `hotHalo` component class is inactive for differential equation solving.
Code lines: 12
Contained by: module galacticus_nodes

subroutine: `hothalocoldmodemasscoldrate`
Description: Accumulate to the rate of change of the `massCold` property of an `coldMode` implementation of the `hotHalo` component class.
Code lines: 31
Contained by: module galacticus_nodes

function: `hothalocoldmodemasscoldrateget`
Description: Get the rate of change of the `massCold` property of an `coldMode` implementation of the `hotHalo` component class.
Code lines: 23
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: `hothalocoldmodemasscoldscale`
Description: Set the absolute scale of the `massCold` property of an `coldMode` implementation of the `hotHalo` component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: `hothalocoldmodemasscoldset`
Description: Set the `massCold` property of an `coldMode` implementation of the `hotHalo` component class.
Code lines: 10

Contained by: module galacticus_nodes

function: hothalocoldmodeouterradiusgrowthrate
Description: Call the deferred function for the outerRadiusGrowthRate method of the hotHalo component class if it has been set.
Code lines: 21
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: hothalocoldmodeouterradiusgrowthratedeferredfunctionset
Description: Set the function to be used for the outerRadiusGrowthRate method of the coldMode implementation of the hotHalo component class.
Code lines: 10
Contained by: module galacticus_nodes

function: hothalocoldmodeouterradiusgrowthratedfrrdfnctnisset
Description: Return true if the deferred function for the outerRadiusGrowthRate method of the coldMode implementation of the hotHalo component class has been set.
Code lines: 8
Contained by: module galacticus_nodes

subroutine: hothalocoldmodeoutflowreturn
Description: Call the deferred function for the outflowReturn method of the hotHalo component class if it has been set.
Code lines: 22
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: hothalocoldmodeoutflowreturndeferredfunctionset
Description: Set the function to be used for the outflowReturn method of the coldMode implementation of the hotHalo component class.
Code lines: 10
Contained by: module galacticus_nodes

function: hothalocoldmodeoutflowreturndfrrdfnctnisset
Description: Return true if the deferred function for the outflowReturn method of the coldMode implementation of the hotHalo component class has been set.
Code lines: 8
Contained by: module galacticus_nodes

subroutine: hothalocoldmodeoutputcount
Description: Increment the count of properties to output for a coldMode implementation of the hotHalo component.
Code lines: 16
Contained by: module galacticus_nodes

subroutine: hothalocoldmodeoutputnames
Description: Return the names of properties to output for a coldMode implementation of the hotHalo component.
Code lines: 34
Contained by: module galacticus_nodes

subroutine: `hothalocoldmodepostoutput`
Description: Perform post-output processing for a `coldMode` implementation of the `hotHalo` component.
Code lines: 11
Contained by: module galacticus_nodes

subroutine: `hothalocreatebyinterrupt`
Description: Create the `hotHalo` component of `self` via an interrupt.
Code lines: 14
Contained by: module galacticus_nodes

function: `hothaloheatsource`
Description: Returns the default value for the `heatSource` property for the `hotHalo` component class.
Code lines: 13
Contained by: module galacticus_nodes

function: `hothaloheatsourceattributematch`
Description: Return a text list of component implementations in the `hotHalo` class that have the desired attributes for the `heatSource` property
Code lines: 52
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: `hothaloheatsourceisgettable`
Description: Returns true if the `heatSource` property is gettable for the `hotHalo` component class.
Code lines: 8
Contained by: module galacticus_nodes

function: `hothaloheatsourcerateget`
Description: Returns a zero rate for the `heatSource` property for the `hotHalo` component class.
Code lines: 13
Contained by: module galacticus_nodes

function: `hothalohothalocoolingabundances`
Description: Returns the default value for the `hotHaloCoolingAbundances` property for the `hotHalo` component class.
Code lines: 13
Contained by: module galacticus_nodes

function: `hothalohothalocoolingabundancesattributematch`
Description: Return a text list of component implementations in the `hotHalo` class that have the desired attributes for the `hotHaloCoolingAbundances` property
Code lines: 74
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: `hothalohothalocoolingabundancesisgettable`
Description: Returns true if the `hotHaloCoolingAbundances` property is gettable for the `hotHalo` component class.
Code lines: 8

Contained by: module galacticus_nodes

subroutine: hothalohothalocoolingabundancesratefunction
Description: Set the function to be used for the rate method of the hotHaloCoolingAbundances property of the hotHalo component.
Code lines: 10
Contained by: module galacticus_nodes

function: hothalohothalocoolingabundancesrateget
Description: Returns a zero rate for the hotHaloCoolingAbundances property for the hotHalo component class.
Code lines: 13
Contained by: module galacticus_nodes

function: hothalohothalocoolingabundancesrateisattached
Description: Return true if the deferred function used to rate the hotHaloCoolingAbundances property of the hotHalo component class has been attached.
Code lines: 8
Contained by: module galacticus_nodes

function: hothalohothalocoolingangularmomentum
Description: Returns the default value for the hotHaloCoolingAngularMomentum property for the hotHalo component class.
Code lines: 13
Contained by: module galacticus_nodes

function: hothalohothalocoolingangularmomentumattributematch
Description: Return a text list of component implementations in the hotHalo class that have the desired attributes for the hotHaloCoolingAngularMomentum property
Code lines: 52
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: hothalohothalocoolingangularmomentumisgettable
Description: Returns true if the hotHaloCoolingAngularMomentum property is gettable for the hotHalo component class.
Code lines: 8
Contained by: module galacticus_nodes

subroutine: hothalohothalocoolingangularmomentumratefunction
Description: Set the function to be used for the rate method of the hotHaloCoolingAngularMomentum property of the hotHalo component.
Code lines: 10
Contained by: module galacticus_nodes

function: hothalohothalocoolingangularmomentumrateget
Description: Returns a zero rate for the hotHaloCoolingAngularMomentum property for the hotHalo component class.
Code lines: 13
Contained by: module galacticus_nodes

function: `hothalohothalocoolingangularmomentumrateisattached`
Description: Return true if the deferred function used to rate the `hotHaloCoolingAngularMomentum` property of the `hotHalo` component class has been attached.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `hothalohothalocoolingmass`
Description: Returns the default value for the `hotHaloCoolingMass` property for the `hotHalo` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `hothalohothalocoolingmassattributematch`
Description: Return a text list of component implementations in the `hotHalo` class that have the desired attributes for the `hotHaloCoolingMass` property
Code lines: 74
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `hothalohothalocoolingmassisgettable`
Description: Returns true if the `hotHaloCoolingMass` property is gettable for the `hotHalo` component class.
Code lines: 8
Contained by: module `galacticus_nodes`

subroutine: `hothalohothalocoolingmassratefunction`
Description: Set the function to be used for the `rate` method of the `hotHaloCoolingMass` property of the `hotHalo` component.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `hothalohothalocoolingmassrateget`
Description: Returns a zero rate for the `hotHaloCoolingMass` property for the `hotHalo` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `hothalohothalocoolingmassrateisattached`
Description: Return true if the deferred function used to rate the `hotHaloCoolingMass` property of the `hotHalo` component class has been attached.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `hothaloisinitialized`
Description: Returns the default value for the `isInitialized` property for the `hotHalo` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `hothaloisinitializedattributematch`
Description: Return a text list of component implementations in the `hotHalo` class that have the desired attributes for the `isInitialized` property

Code lines: 52
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: hothaloisinitializedisgettable
Description: Returns true if the isInitialized property is gettable for the hotHalo component class.
Code lines: 8
Contained by: module galacticus_nodes

function: hothaloisinitializedrateget
Description: Returns a zero rate for the isInitialized property for the hotHalo component class.
Code lines: 13
Contained by: module galacticus_nodes

function: hothalomass
Description: Returns the default value for the mass property for the hotHalo component class.
Code lines: 13
Contained by: module galacticus_nodes

function: hothalomassattributematch
Description: Return a text list of component implementations in the hotHalo class that have the desired attributes for the mass property
Code lines: 64
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: hothalomasscold
Description: Returns the default value for the massCold property for the hotHalo component class.
Code lines: 13
Contained by: module galacticus_nodes

function: hothalomasscoldattributematch
Description: Return a text list of component implementations in the hotHalo class that have the desired attributes for the massCold property
Code lines: 28
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: hothalomasscoldisgettable
Description: Returns true if the massCold property is gettable for the hotHalo component class.
Code lines: 8
Contained by: module galacticus_nodes

subroutine: hothalomasscoldrate
Description: Accept a rate set for the massCold property of the hotHalo component class. Trigger an interrupt to create the component.
Code lines: 21
Contained by: module galacticus_nodes
Modules used: galacticus_error

function: `hothalomasscoldrateget`
Description: Returns a zero rate for the `massCold` property for the `hotHalo` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `hothalomassisgettable`
Description: Returns true if the `mass` property is gettable for the `hotHalo` component class.
Code lines: 8
Contained by: module `galacticus_nodes`

subroutine: `hothalomassrate`
Description: Accept a rate set for the `mass` property of the `hotHalo` component class. Trigger an interrupt to create the component.
Code lines: 21
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

function: `hothalomassrateget`
Description: Returns a zero rate for the `mass` property for the `hotHalo` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `hothalomasssink`
Description: Returns the default value for the `massSink` property for the `hotHalo` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `hothalomasssinkattributematch`
Description: Return a text list of component implementations in the `hotHalo` class that have the desired attributes for the `massSink` property
Code lines: 52
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `hothalomasssinkisgettable`
Description: Returns true if the `massSink` property is gettable for the `hotHalo` component class.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `hothalomasssinkrateget`
Description: Returns a zero rate for the `massSink` property for the `hotHalo` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `hothalomasstotal`
Description: Returns the default value for the `massTotal` property for the `hotHalo` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `hothalomasstotalattributematch`
Description: Return a text list of component implementations in the `hotHalo` class that have the desired attributes for the `massTotal` property
Code lines: 52
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `hothalomasstotalisgettable`
Description: Returns true if the `massTotal` property is gettable for the `hotHalo` component class.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `hothalomasstotalrateget`
Description: Returns a zero rate for the `massTotal` property for the `hotHalo` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

subroutine: `hothalonulloutputcount`
Description: Increment the count of properties to output for a `null` implementation of the `hotHalo` component.
Code lines: 22
Contained by: module `galacticus_nodes`

subroutine: `hothalonulloutputnames`
Description: Return the names of properties to output for a `null` implementation of the `hotHalo` component.
Code lines: 39
Contained by: module `galacticus_nodes`

function: `hothaloouterradius`
Description: Returns the default value for the `outerRadius` property for the `hotHalo` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `hothaloouterradiusattributematch`
Description: Return a text list of component implementations in the `hotHalo` class that have the desired attributes for the `outerRadius` property
Code lines: 68
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `hothaloouterradiusisgettable`
Description: Returns true if the `outerRadius` property is gettable for the `hotHalo` component class.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `hothaloouterradiusrateget`
Description: Returns a zero rate for the `outerRadius` property for the `hotHalo` component class.
Code lines: 13

Contained by: module galacticus_nodes

function: hothalooutflowedabundances
Description: Returns the default value for the outflowedAbundances property for the hotHalo component class.
Code lines: 13
Contained by: module galacticus_nodes

function: hothalooutflowedabundancesattributematch
Description: Return a text list of component implementations in the hotHalo class that have the desired attributes for the outflowedAbundances property
Code lines: 55
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: hothalooutflowedabundancesisgettable
Description: Returns true if the outflowedAbundances property is gettable for the hotHalo component class.
Code lines: 8
Contained by: module galacticus_nodes

function: hothalooutflowedabundancesrateget
Description: Returns a zero rate for the outflowedAbundances property for the hotHalo component class.
Code lines: 13
Contained by: module galacticus_nodes

function: hothalooutflowedangularmomentum
Description: Returns the default value for the outflowedAngularMomentum property for the hotHalo component class.
Code lines: 13
Contained by: module galacticus_nodes

function: hothalooutflowedangularmomentumattributematch
Description: Return a text list of component implementations in the hotHalo class that have the desired attributes for the outflowedAngularMomentum property
Code lines: 46
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: hothalooutflowedangularmomentumisgettable
Description: Returns true if the outflowedAngularMomentum property is gettable for the hotHalo component class.
Code lines: 8
Contained by: module galacticus_nodes

function: hothalooutflowedangularmomentumrateget
Description: Returns a zero rate for the outflowedAngularMomentum property for the hotHalo component class.
Code lines: 13
Contained by: module galacticus_nodes

function: `hothalooutflowedmass`
Description: Returns the default value for the `outflowedMass` property for the `hotHalo` component class.
Code lines: 13
Contained by: module galacticus_nodes

function: `hothalooutflowedmassattributematch`
Description: Return a text list of component implementations in the `hotHalo` class that have the desired attributes for the `outflowedMass` property
Code lines: 55
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: `hothalooutflowedmassisgettable`
Description: Returns true if the `outflowedMass` property is gettable for the `hotHalo` component class.
Code lines: 8
Contained by: module galacticus_nodes

function: `hothalooutflowedmassrateget`
Description: Returns a zero rate for the `outflowedMass` property for the `hotHalo` component class.
Code lines: 13
Contained by: module galacticus_nodes

function: `hothalooutflowingabundances`
Description: Returns the default value for the `outflowingAbundances` property for the `hotHalo` component class.
Code lines: 13
Contained by: module galacticus_nodes

function: `hothalooutflowingabundancesattributematch`
Description: Return a text list of component implementations in the `hotHalo` class that have the desired attributes for the `outflowingAbundances` property
Code lines: 74
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: `hothalooutflowingabundancesisgettable`
Description: Returns true if the `outflowingAbundances` property is gettable for the `hotHalo` component class.
Code lines: 8
Contained by: module galacticus_nodes

function: `hothalooutflowingabundancesrateget`
Description: Returns a zero rate for the `outflowingAbundances` property for the `hotHalo` component class.
Code lines: 13
Contained by: module galacticus_nodes

function: `hothalooutflowingangularmomentum`
Description: Returns the default value for the `outflowingAngularMomentum` property for the `hotHalo` component class.

Code lines: 13
Contained by: module galacticus_nodes

function: hothalooutflowingangularmomentumattributematch
Description: Return a text list of component implementations in the hotHalo class that have the desired attributes for the outflowingAngularMomentum property
Code lines: 52
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: hothalooutflowingangularmomentumisgettable
Description: Returns true if the outflowingAngularMomentum property is gettable for the hotHalo component class.
Code lines: 8
Contained by: module galacticus_nodes

function: hothalooutflowingangularmomentumrateget
Description: Returns a zero rate for the outflowingAngularMomentum property for the hotHalo component class.
Code lines: 13
Contained by: module galacticus_nodes

function: hothalooutflowingmass
Description: Returns the default value for the outflowingMass property for the hotHalo component class.
Code lines: 13
Contained by: module galacticus_nodes

function: hothalooutflowingmassattributematch
Description: Return a text list of component implementations in the hotHalo class that have the desired attributes for the outflowingMass property
Code lines: 74
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: hothalooutflowingmassisgettable
Description: Returns true if the outflowingMass property is gettable for the hotHalo component class.
Code lines: 8
Contained by: module galacticus_nodes

function: hothalooutflowingmassrateget
Description: Returns a zero rate for the outflowingMass property for the hotHalo component class.
Code lines: 13
Contained by: module galacticus_nodes

function: hothalooutflowtrackingouterradiusgrowthrate
Description: Call the deferred function for the outerRadiusGrowthRate method of the hotHalo component class if it has been set.
Code lines: 21
Contained by: module galacticus_nodes

Modules used: galacticus_error

subroutine: hothalooutflowtrackingouterradiusgrowthratedeferredfunctionset
Description: Set the function to be used for the outerRadiusGrowthRate method of the outflowTracking implementation of the hotHalo component class.
Code lines: 10
Contained by: module galacticus_nodes

function: hothalooutflowtrackingouterradiusgrowthratedfrrdfnctnisset
Description: Return true if the deferred function for the outerRadiusGrowthRate method of the outflowTracking implementation of the hotHalo component class has been set.
Code lines: 8
Contained by: module galacticus_nodes

subroutine: hothalooutflowtrackingoutflowreturn
Description: Call the deferred function for the outflowReturn method of the hotHalo component class if it has been set.
Code lines: 22
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: hothalooutflowtrackingoutflowreturndeferredfunctionset
Description: Set the function to be used for the outflowReturn method of the outflowTracking implementation of the hotHalo component class.
Code lines: 10
Contained by: module galacticus_nodes

function: hothalooutflowtrackingoutflowreturndfrrdfnctnisset
Description: Return true if the deferred function for the outflowReturn method of the outflowTracking implementation of the hotHalo component class has been set.
Code lines: 8
Contained by: module galacticus_nodes

subroutine: hothalooutflowtrackingoutputcount
Description: Increment the count of properties to output for a outflowTracking implementation of the hotHalo component.
Code lines: 16
Contained by: module galacticus_nodes

subroutine: hothalooutflowtrackingoutputnames
Description: Return the names of properties to output for a outflowTracking implementation of the hotHalo component.
Code lines: 28
Contained by: module galacticus_nodes

subroutine: hothalooutflowtrackingpostoutput
Description: Perform post-output processing for a outflowTracking implementation of the hotHalo component.
Code lines: 11
Contained by: module galacticus_nodes

function: `hothalooutflowtrackingtrackedoutflowabundancescount`
Description: Return a count of the number of scalar properties in the `trackedOutflowAbundances` property of an `outflowTracking` implementation of the `hotHalo` component class.
Code lines: 12
Contained by: module galacticus_nodes

function: `hothalooutflowtrackingtrackedoutflowabundancesget`
Description: Get the `trackedOutflowAbundances` property of an `outflowTracking` implementation of the `hotHalo` component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: `hothalooutflowtrackingtrackedoutflowabundancesjcbnzr`
Description: Indicate that the `trackedOutflowAbundances` property of an `outflowTracking` implementation of the `hotHalo` component class is inactive for differential equation solving.
Code lines: 14
Contained by: module galacticus_nodes

subroutine: `hothalooutflowtrackingtrackedoutflowabundancesrate`
Description: Accumulate to the rate of change of the `trackedOutflowAbundances` property of an `outflowTracking` implementation of the `hotHalo` component class.
Code lines: 38
Contained by: module galacticus_nodes

function: `hothalooutflowtrackingtrackedoutflowabundancesrateget`
Description: Get the rate of change of the `trackedOutflowAbundances` property of an `outflowTracking` implementation of the `hotHalo` component class.
Code lines: 26
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: `hothalooutflowtrackingtrackedoutflowabundancesscale`
Description: Set the absolute scale of the `trackedOutflowAbundances` property of an `outflowTracking` implementation of the `hotHalo` component class.
Code lines: 15
Contained by: module galacticus_nodes

subroutine: `hothalooutflowtrackingtrackedoutflowabundancesset`
Description: Set the `trackedOutflowAbundances` property of an `outflowTracking` implementation of the `hotHalo` component class.
Code lines: 10
Contained by: module galacticus_nodes

function: `hothalooutflowtrackingtrackedoutflowmasscount`
Description: Return a count of the number of scalar properties in the `trackedOutflowMass` property of an `outflowTracking` implementation of the `hotHalo` component class.
Code lines: 12
Contained by: module galacticus_nodes

function: `hothalooutflowtrackingtrackedoutflowmassget`
Description: Get the `trackedOutflowMass` property of an `outflowTracking` implementation of the `hotHalo` component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: `hothalooutflowtrackingtrackedoutflowmassjcbnzr`
Description: Indicate that the `trackedOutflowMass` property of an `outflowTracking` implementation of the `hotHalo` component class is inactive for differential equation solving.
Code lines: 12
Contained by: module galacticus_nodes

subroutine: `hothalooutflowtrackingtrackedoutflowmassrate`
Description: Accumulate to the rate of change of the `trackedOutflowMass` property of an `outflowTracking` implementation of the `hotHalo` component class.
Code lines: 31
Contained by: module galacticus_nodes

function: `hothalooutflowtrackingtrackedoutflowmassrateget`
Description: Get the rate of change of the `trackedOutflowMass` property of an `outflowTracking` implementation of the `hotHalo` component class.
Code lines: 23
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: `hothalooutflowtrackingtrackedoutflowmassscale`
Description: Set the absolute scale of the `trackedOutflowMass` property of an `outflowTracking` implementation of the `hotHalo` component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: `hothalooutflowtrackingtrackedoutflowmassset`
Description: Set the `trackedOutflowMass` property of an `outflowTracking` implementation of the `hotHalo` component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: `hothalooutput`
Description: Populate output buffers with properties to output for a `hotHalo` component.
Code lines: 75
Contained by: module galacticus_nodes
Modules used: multi_counters

subroutine: `hothalooutputcount`
Description: Increment the count of properties to output for a generic `hotHalo` component.
Code lines: 15
Contained by: module galacticus_nodes

subroutine: `hothalooutputnames`
Description: Establish the names of properties to output for a generic `hotHalo` component.
Code lines: 20
Contained by: module `galacticus_nodes`

subroutine: `hothalopostoutput`
Description: Perform post-output processing of a `hotHalo` component.
Code lines: 14
Contained by: module `galacticus_nodes`

function: `hothalostandardabundancescount`
Description: Return a count of the number of scalar properties in the `abundances` property of an `standard` implementation of the `hotHalo` component class.
Code lines: 12
Contained by: module `galacticus_nodes`

function: `hothalostandardabundancesget`
Description: Get the `abundances` property of an `standard` implementation of the `hotHalo` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `hothalostandardabundancesjcbnzr`
Description: Indicate that the `abundances` property of an `standard` implementation of the `hotHalo` component class is inactive for differential equation solving.
Code lines: 14
Contained by: module `galacticus_nodes`

subroutine: `hothalostandardabundancesrate`
Description: Accumulate to the rate of change of the `abundances` property of an `standard` implementation of the `hotHalo` component class.
Code lines: 38
Contained by: module `galacticus_nodes`

function: `hothalostandardabundancesrateget`
Description: Get the rate of change of the `abundances` property of an `standard` implementation of the `hotHalo` component class.
Code lines: 26
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `hothalostandardabundancesscale`
Description: Set the absolute scale of the `abundances` property of an `standard` implementation of the `hotHalo` component class.
Code lines: 15
Contained by: module `galacticus_nodes`

subroutine: `hothalostandardabundancesset`
Description: Set the `abundances` property of an `standard` implementation of the `hotHalo` component class.

Code lines: 10
Contained by: module galacticus_nodes

function: hothalostandardangularmomentumcount
Description: Return a count of the number of scalar properties in the angularMomentum property of an standard implementation of the hotHalo component class.
Code lines: 12
Contained by: module galacticus_nodes

function: hothalostandardangularmomentumget
Description: Get the angularMomentum property of an standard implementation of the hotHalo component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: hothalostandardangularmomentumjcbnzr
Description: Indicate that the angularMomentum property of an standard implementation of the hotHalo component class is inactive for differential equation solving.
Code lines: 12
Contained by: module galacticus_nodes

subroutine: hothalostandardangularmomentumrate
Description: Accumulate to the rate of change of the angularMomentum property of an standard implementation of the hotHalo component class.
Code lines: 31
Contained by: module galacticus_nodes

function: hothalostandardangularmomentumrateget
Description: Get the rate of change of the angularMomentum property of an standard implementation of the hotHalo component class.
Code lines: 23
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: hothalostandardangularmomentumscale
Description: Set the absolute scale of the angularMomentum property of an standard implementation of the hotHalo component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: hothalostandardangularmomentumset
Description: Set the angularMomentum property of an standard implementation of the hotHalo component class.
Code lines: 10
Contained by: module galacticus_nodes

function: hothalostandardchemicalscount
Description: Return a count of the number of scalar properties in the chemicals property of an standard implementation of the hotHalo component class.
Code lines: 12
Contained by: module galacticus_nodes

function: `hothalostandardchemicalsget`
Description: Get the `chemicals` property of an `standard` implementation of the `hotHalo` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `hothalostandardchemicalsjcbnzr`
Description: Indicate that the `chemicals` property of an `standard` implementation of the `hotHalo` component class is inactive for differential equation solving.
Code lines: 14
Contained by: module `galacticus_nodes`

subroutine: `hothalostandardchemicalsrate`
Description: Accumulate to the rate of change of the `chemicals` property of an `standard` implementation of the `hotHalo` component class.
Code lines: 38
Contained by: module `galacticus_nodes`

function: `hothalostandardchemicalsrateget`
Description: Get the rate of change of the `chemicals` property of an `standard` implementation of the `hotHalo` component class.
Code lines: 26
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `hothalostandardchemicalsscale`
Description: Set the absolute scale of the `chemicals` property of an `standard` implementation of the `hotHalo` component class.
Code lines: 15
Contained by: module `galacticus_nodes`

subroutine: `hothalostandardchemicalsset`
Description: Set the `chemicals` property of an `standard` implementation of the `hotHalo` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `hothalostandardcreatefunctionset`
Description: Set the create function for the `standard` implementation of the `hotHalo` component class.
Code lines: 9
Contained by: module `galacticus_nodes`

subroutine: `hothalostandardheatsourcerate`
Description: Accumulate the rate of change of the `heatSource` property of the `standard` implementation of the `hotHalo` component using a deferred function.
Code lines: 12
Contained by: module `galacticus_nodes`

subroutine: `hothalostandardheatsourceratefunction`
Description: Set the function to be used for the `rate` method of the `heatSource` property of the `HotHaloStandard` component.

Code lines: 10
Contained by: module galacticus_nodes

function: hothalostandardheatsourcerateisattached
Description: Return true if the deferred function used to rate the heatSource property of the HotHaloStandard component class has been attached.
Code lines: 8
Contained by: module galacticus_nodes

subroutine: hothalostandardhothalocoolingabundancesrate
Description: Accumulate the rate of change of the hotHaloCoolingAbundances property of the standard implementation of the hotHalo component using a deferred function.
Code lines: 12
Contained by: module galacticus_nodes

subroutine: hothalostandardhothalocoolingangularmomentumrate
Description: Accumulate the rate of change of the hotHaloCoolingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.
Code lines: 12
Contained by: module galacticus_nodes

subroutine: hothalostandardhothalocoolingmassrate
Description: Accumulate the rate of change of the hotHaloCoolingMass property of the standard implementation of the hotHalo component using a deferred function.
Code lines: 12
Contained by: module galacticus_nodes

function: hothalostandardisinitializedget
Description: Get the isInitialized property of an standard implementation of the hotHalo component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: hothalostandardisinitializedset
Description: Set the isInitialized property of an standard implementation of the hotHalo component class.
Code lines: 10
Contained by: module galacticus_nodes

function: hothalostandardmasscount
Description: Return a count of the number of scalar properties in the mass property of an standard implementation of the hotHalo component class.
Code lines: 12
Contained by: module galacticus_nodes

function: hothalostandardmassget
Description: Get the mass property of an standard implementation of the hotHalo component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: `hothalostandardmassjcbnzr`
Description: Indicate that the `mass` property of an `standard` implementation of the `hotHalo` component class is inactive for differential equation solving.
Code lines: 12
Contained by: module `galacticus_nodes`

subroutine: `hothalostandardmassrate`
Description: Accumulate to the rate of change of the `mass` property of an `standard` implementation of the `hotHalo` component class.
Code lines: 31
Contained by: module `galacticus_nodes`

function: `hothalostandardmassrateget`
Description: Get the rate of change of the `mass` property of an `standard` implementation of the `hotHalo` component class.
Code lines: 23
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `hothalostandardmassscale`
Description: Set the absolute scale of the `mass` property of an `standard` implementation of the `hotHalo` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

subroutine: `hothalostandardmassset`
Description: Set the `mass` property of an `standard` implementation of the `hotHalo` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `hothalostandardmasssinkrate`
Description: Accumulate the rate of change of the `massSink` property of the `standard` implementation of the `hotHalo` component using a deferred function.
Code lines: 12
Contained by: module `galacticus_nodes`

subroutine: `hothalostandardmasssinkratefunction`
Description: Set the function to be used for the `rate` method of the `massSink` property of the `HotHaloStandard` component.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `hothalostandardmasssinkrateisattached`
Description: Return true if the deferred function used to rate the `massSink` property of the `HotHaloStandard` component class has been attached.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `hothalostandardouterradiuscount`
Description: Return a count of the number of scalar properties in the `outerRadius` property of an `standard` implementation of the `hotHalo` component class.

Code lines: 12
Contained by: module galacticus_nodes

function: hothalostandardouterradiusget
Description: Get the value of the outerRadius property of the standard implementation of the hotHalo component using a deferred function.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: hothalostandardouterradiusgetfunction
Description: Set the function to be used for the get method of the outerRadius property of the HotHaloStandard component.
Code lines: 10
Contained by: module galacticus_nodes

function: hothalostandardouterradiusgetisattached
Description: Return true if the deferred function used to get the outerRadius property of the HotHaloStandard component class has been attached.
Code lines: 8
Contained by: module galacticus_nodes

function: hothalostandardouterradiusgetvalue
Description: Get the outerRadius property of an standard implementation of the hotHalo component class.
Code lines: 10
Contained by: module galacticus_nodes

function: hothalostandardouterradiusgrowthrate
Description: Call the deferred function for the outerRadiusGrowthRate method of the hotHalo component class if it has been set.
Code lines: 22
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: hothalostandardouterradiusgrowthratedeferredfunctionset
Description: Set the function to be used for the outerRadiusGrowthRate method of the standard implementation of the hotHalo component class.
Code lines: 10
Contained by: module galacticus_nodes

function: hothalostandardouterradiusgrowthratedfrrdfnctnisset
Description: Return true if the deferred function for the outerRadiusGrowthRate method of the standard implementation of the hotHalo component class has been set.
Code lines: 8
Contained by: module galacticus_nodes

subroutine: hothalostandardouterradiusjcbnzr
Description: Indicate that the outerRadius property of an standard implementation of the hotHalo component class is inactive for differential equation solving.
Code lines: 12
Contained by: module galacticus_nodes

subroutine: `hothalostandardouterradiusrate`
Description: Accumulate to the rate of change of the `outerRadius` property of an `standard` implementation of the `hotHalo` component class.
Code lines: 31
Contained by: module `galacticus_nodes`

function: `hothalostandardouterradiusrateget`
Description: Get the rate of change of the `outerRadius` property of an `standard` implementation of the `hotHalo` component class.
Code lines: 23
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `hothalostandardouterradiusscale`
Description: Set the absolute scale of the `outerRadius` property of an `standard` implementation of the `hotHalo` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

subroutine: `hothalostandardouterradiusset`
Description: Set the `outerRadius` property of an `standard` implementation of the `hotHalo` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `hothalostandardoutflowedabundancescount`
Description: Return a count of the number of scalar properties in the `outflowedAbundances` property of an `standard` implementation of the `hotHalo` component class.
Code lines: 12
Contained by: module `galacticus_nodes`

function: `hothalostandardoutflowedabundancesget`
Description: Get the `outflowedAbundances` property of an `standard` implementation of the `hotHalo` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `hothalostandardoutflowedabundancesjcbnzr`
Description: Indicate that the `outflowedAbundances` property of an `standard` implementation of the `hotHalo` component class is inactive for differential equation solving.
Code lines: 14
Contained by: module `galacticus_nodes`

subroutine: `hothalostandardoutflowedabundancesrate`
Description: Accumulate to the rate of change of the `outflowedAbundances` property of an `standard` implementation of the `hotHalo` component class.
Code lines: 38
Contained by: module `galacticus_nodes`

function: `hothalostandardoutflowedabundancesrateget`

Description: Get the rate of change of the `outflowedAbundances` property of an `standard` implementation of the `hotHalo` component class.
Code lines: 26
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: `hothalostandardoutflowedabundancesscale`
Description: Set the absolute scale of the `outflowedAbundances` property of an `standard` implementation of the `hotHalo` component class.
Code lines: 15
Contained by: module galacticus_nodes

subroutine: `hothalostandardoutflowedabundancesset`
Description: Set the `outflowedAbundances` property of an `standard` implementation of the `hotHalo` component class.
Code lines: 10
Contained by: module galacticus_nodes

function: `hothalostandardoutflowedangularmomentumcount`
Description: Return a count of the number of scalar properties in the `outflowedAngularMomentum` property of an `standard` implementation of the `hotHalo` component class.
Code lines: 12
Contained by: module galacticus_nodes

function: `hothalostandardoutflowedangularmomentumget`
Description: Get the `outflowedAngularMomentum` property of an `standard` implementation of the `hotHalo` component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: `hothalostandardoutflowedangularmomentumjcbnzr`
Description: Indicate that the `outflowedAngularMomentum` property of an `standard` implementation of the `hotHalo` component class is inactive for differential equation solving.
Code lines: 12
Contained by: module galacticus_nodes

subroutine: `hothalostandardoutflowedangularmomentumrate`
Description: Accumulate to the rate of change of the `outflowedAngularMomentum` property of an `standard` implementation of the `hotHalo` component class.
Code lines: 31
Contained by: module galacticus_nodes

function: `hothalostandardoutflowedangularmomentumrateget`
Description: Get the rate of change of the `outflowedAngularMomentum` property of an `standard` implementation of the `hotHalo` component class.
Code lines: 23
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: `hothalostandardoutflowedangularmomentumscale`

Description: Set the absolute scale of the `outflowedAngularMomentum` property of an `standard` implementation of the `hotHalo` component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: `hothalostandardoutflowedangularmomentumset`
Description: Set the `outflowedAngularMomentum` property of an `standard` implementation of the `hotHalo` component class.
Code lines: 10
Contained by: module galacticus_nodes

function: `hothalostandardoutflowedmasscount`
Description: Return a count of the number of scalar properties in the `outflowedMass` property of an `standard` implementation of the `hotHalo` component class.
Code lines: 12
Contained by: module galacticus_nodes

function: `hothalostandardoutflowedmassget`
Description: Get the `outflowedMass` property of an `standard` implementation of the `hotHalo` component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: `hothalostandardoutflowedmassjcbnzr`
Description: Indicate that the `outflowedMass` property of an `standard` implementation of the `hotHalo` component class is inactive for differential equation solving.
Code lines: 12
Contained by: module galacticus_nodes

subroutine: `hothalostandardoutflowedmassrate`
Description: Accumulate to the rate of change of the `outflowedMass` property of an `standard` implementation of the `hotHalo` component class.
Code lines: 31
Contained by: module galacticus_nodes

function: `hothalostandardoutflowedmassrateget`
Description: Get the rate of change of the `outflowedMass` property of an `standard` implementation of the `hotHalo` component class.
Code lines: 23
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: `hothalostandardoutflowedmassscale`
Description: Set the absolute scale of the `outflowedMass` property of an `standard` implementation of the `hotHalo` component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: `hothalostandardoutflowedmassset`
Description: Set the `outflowedMass` property of an `standard` implementation of the `hotHalo` component class.
Code lines: 10

Contained by: module galacticus_nodes

subroutine: hothalostandardoutflowingabundancesrate
Description: Accumulate the rate of change of the outflowingAbundances property of the standard implementation of the hotHalo component using a deferred function.
Code lines: 12
Contained by: module galacticus_nodes

subroutine: hothalostandardoutflowingabundancesratefunction
Description: Set the function to be used for the rate method of the outflowingAbundances property of the HotHaloStandard component.
Code lines: 10
Contained by: module galacticus_nodes

function: hothalostandardoutflowingabundancesrateisattached
Description: Return true if the deferred function used to rate the outflowingAbundances property of the HotHaloStandard component class has been attached.
Code lines: 8
Contained by: module galacticus_nodes

subroutine: hothalostandardoutflowingangularmomentumrate
Description: Accumulate the rate of change of the outflowingAngularMomentum property of the standard implementation of the hotHalo component using a deferred function.
Code lines: 12
Contained by: module galacticus_nodes

subroutine: hothalostandardoutflowingangularmomentumratefunction
Description: Set the function to be used for the rate method of the outflowingAngularMomentum property of the HotHaloStandard component.
Code lines: 10
Contained by: module galacticus_nodes

function: hothalostandardoutflowingangularmomentumrateisattached
Description: Return true if the deferred function used to rate the outflowingAngularMomentum property of the HotHaloStandard component class has been attached.
Code lines: 8
Contained by: module galacticus_nodes

subroutine: hothalostandardoutflowingmassrate
Description: Accumulate the rate of change of the outflowingMass property of the standard implementation of the hotHalo component using a deferred function.
Code lines: 12
Contained by: module galacticus_nodes

subroutine: hothalostandardoutflowingmassratefunction
Description: Set the function to be used for the rate method of the outflowingMass property of the HotHaloStandard component.
Code lines: 10
Contained by: module galacticus_nodes

function: `hothalostandardoutflowingmassrateisattached`
Description: Return true if the deferred function used to rate the `outflowingMass` property of the `HotHaloStandard` component class has been attached.
Code lines: 8
Contained by: module galacticus_nodes

subroutine: `hothalostandardoutflowreturn`
Description: Call the deferred function for the `outflowReturn` method of the `hotHalo` component class if it has been set.
Code lines: 22
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: `hothalostandardoutflowreturndeferredfunctionset`
Description: Set the function to be used for the `outflowReturn` method of the `standard` implementation of the `hotHalo` component class.
Code lines: 10
Contained by: module galacticus_nodes

function: `hothalostandardoutflowreturndfrrdfnctnisset`
Description: Return true if the deferred function for the `outflowReturn` method of the `standard` implementation of the `hotHalo` component class has been set.
Code lines: 8
Contained by: module galacticus_nodes

subroutine: `hothalostandardoutputcount`
Description: Increment the count of properties to output for a `standard` implementation of the `hotHalo` component.
Code lines: 20
Contained by: module galacticus_nodes

subroutine: `hothalostandardoutputnames`
Description: Return the names of properties to output for a `standard` implementation of the `hotHalo` component.
Code lines: 64
Contained by: module galacticus_nodes

subroutine: `hothalostandardpostoutput`
Description: Perform post-output processing for a `standard` implementation of the `hotHalo` component.
Code lines: 11
Contained by: module galacticus_nodes

function: `hothalostandardstrippedabundancescount`
Description: Return a count of the number of scalar properties in the `strippedAbundances` property of an `standard` implementation of the `hotHalo` component class.
Code lines: 12
Contained by: module galacticus_nodes

function: `hothalostandardstrippedabundancesget`

Description: Get the `strippedAbundances` property of an `standard` implementation of the `hotHalo` component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: `hothalostandardstrippedabundancesjcbnzr`
Description: Indicate that the `strippedAbundances` property of an `standard` implementation of the `hotHalo` component class is inactive for differential equation solving.
Code lines: 14
Contained by: module galacticus_nodes

subroutine: `hothalostandardstrippedabundancesrate`
Description: Accumulate to the rate of change of the `strippedAbundances` property of an `standard` implementation of the `hotHalo` component class.
Code lines: 38
Contained by: module galacticus_nodes

function: `hothalostandardstrippedabundancesrateget`
Description: Get the rate of change of the `strippedAbundances` property of an `standard` implementation of the `hotHalo` component class.
Code lines: 26
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: `hothalostandardstrippedabundancesscale`
Description: Set the absolute scale of the `strippedAbundances` property of an `standard` implementation of the `hotHalo` component class.
Code lines: 15
Contained by: module galacticus_nodes

subroutine: `hothalostandardstrippedabundancesset`
Description: Set the `strippedAbundances` property of an `standard` implementation of the `hotHalo` component class.
Code lines: 10
Contained by: module galacticus_nodes

function: `hothalostandardstrippedmasscount`
Description: Return a count of the number of scalar properties in the `strippedMass` property of an `standard` implementation of the `hotHalo` component class.
Code lines: 12
Contained by: module galacticus_nodes

function: `hothalostandardstrippedmassget`
Description: Get the `strippedMass` property of an `standard` implementation of the `hotHalo` component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: `hothalostandardstrippedmassjcbnzr`
Description: Indicate that the `strippedMass` property of an `standard` implementation of the `hotHalo` component class is inactive for differential equation solving.

Code lines: 12
Contained by: module galacticus_nodes

subroutine: hothalostandardstrippedmassrate
Description: Accumulate to the rate of change of the strippedMass property of an standard implementation of the hotHalo component class.
Code lines: 31
Contained by: module galacticus_nodes

function: hothalostandardstrippedmassrateget
Description: Get the rate of change of the strippedMass property of an standard implementation of the hotHalo component class.
Code lines: 23
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: hothalostandardstrippedmassscale
Description: Set the absolute scale of the strippedMass property of an standard implementation of the hotHalo component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: hothalostandardstrippedmassset
Description: Set the strippedMass property of an standard implementation of the hotHalo component class.
Code lines: 10
Contained by: module galacticus_nodes

function: hothalostandardunaccretedmasscount
Description: Return a count of the number of scalar properties in the unaccretedMass property of an standard implementation of the hotHalo component class.
Code lines: 12
Contained by: module galacticus_nodes

function: hothalostandardunaccretedmassget
Description: Get the unaccretedMass property of an standard implementation of the hotHalo component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: hothalostandardunaccretedmassjcbnzr
Description: Indicate that the unaccretedMass property of an standard implementation of the hotHalo component class is inactive for differential equation solving.
Code lines: 12
Contained by: module galacticus_nodes

subroutine: hothalostandardunaccretedmassrate
Description: Accumulate to the rate of change of the unaccretedMass property of an standard implementation of the hotHalo component class.
Code lines: 31
Contained by: module galacticus_nodes

function: `hothalostandardunaccretedmassrateget`
Description: Get the rate of change of the `unaccretedMass` property of an `standard` implementation of the `hotHalo` component class.
Code lines: 23
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `hothalostandardunaccretedmassscale`
Description: Set the absolute scale of the `unaccretedMass` property of an `standard` implementation of the `hotHalo` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

subroutine: `hothalostandardunaccretedmassset`
Description: Set the `unaccretedMass` property of an `standard` implementation of the `hotHalo` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `hothalostrippedabundances`
Description: Returns the default value for the `strippedAbundances` property for the `hotHalo` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `hothalostrippedabundancesattributematch`
Description: Return a text list of component implementations in the `hotHalo` class that have the desired attributes for the `strippedAbundances` property
Code lines: 46
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `hothalostrippedabundancesisgettable`
Description: Returns true if the `strippedAbundances` property is gettable for the `hotHalo` component class.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `hothalostrippedabundancesrateget`
Description: Returns a zero rate for the `strippedAbundances` property for the `hotHalo` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `hothalostrippedmass`
Description: Returns the default value for the `strippedMass` property for the `hotHalo` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `hothalostrippedmassattributematch`

Description: Return a text list of component implementations in the `hotHalo` class that have the desired attributes for the `strippedMass` property
Code lines: 46
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: `hothalostrippedmassisgettable`
Description: Returns true if the `strippedMass` property is gettable for the `hotHalo` component class.
Code lines: 8
Contained by: module galacticus_nodes

function: `hothalostrippedmassrateget`
Description: Returns a zero rate for the `strippedMass` property for the `hotHalo` component class.
Code lines: 13
Contained by: module galacticus_nodes

function: `hothalotrackedoutflowabundances`
Description: Returns the default value for the `trackedOutflowAbundances` property for the `hotHalo` component class.
Code lines: 13
Contained by: module galacticus_nodes

function: `hothalotrackedoutflowabundancesattributematch`
Description: Return a text list of component implementations in the `hotHalo` class that have the desired attributes for the `trackedOutflowAbundances` property
Code lines: 28
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: `hothalotrackedoutflowabundancesisgettable`
Description: Returns true if the `trackedOutflowAbundances` property is gettable for the `hotHalo` component class.
Code lines: 8
Contained by: module galacticus_nodes

function: `hothalotrackedoutflowabundancesrateget`
Description: Returns a zero rate for the `trackedOutflowAbundances` property for the `hotHalo` component class.
Code lines: 13
Contained by: module galacticus_nodes

function: `hothalotrackedoutflowmass`
Description: Returns the default value for the `trackedOutflowMass` property for the `hotHalo` component class.
Code lines: 13
Contained by: module galacticus_nodes

function: `hothalotrackedoutflowmassattributematch`
Description: Return a text list of component implementations in the `hotHalo` class that have the desired attributes for the `trackedOutflowMass` property
Code lines: 28

Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: hothalotrackedoutflowmassisgettable
Description: Returns true if the trackedOutflowMass property is gettable for the hotHalo component class.
Code lines: 8
Contained by: module galacticus_nodes

function: hothalotrackedoutflowmassrateget
Description: Returns a zero rate for the trackedOutflowMass property for the hotHalo component class.
Code lines: 13
Contained by: module galacticus_nodes

function: hothalounaccretedmass
Description: Returns the default value for the unaccretedMass property for the hotHalo component class.
Code lines: 13
Contained by: module galacticus_nodes

function: hothalounaccretedmassattributematch
Description: Return a text list of component implementations in the hotHalo class that have the desired attributes for the unaccretedMass property
Code lines: 64
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: hothalounaccretedmassisgettable
Description: Returns true if the unaccretedMass property is gettable for the hotHalo component class.
Code lines: 8
Contained by: module galacticus_nodes

subroutine: hothalounaccretedmassrate
Description: Accept a rate set for the unaccretedMass property of the hotHalo component class. Trigger an interrupt to create the component.
Code lines: 21
Contained by: module galacticus_nodes
Modules used: galacticus_error

function: hothalounaccretedmassrateget
Description: Returns a zero rate for the unaccretedMass property for the hotHalo component class.
Code lines: 13
Contained by: module galacticus_nodes

function: hothaloverysimpleabundancescount
Description: Return a count of the number of scalar properties in the abundances property of an verySimple implementation of the hotHalo component class.
Code lines: 12
Contained by: module galacticus_nodes

function: hothaloverysimpleabundancesget

Description: Get the `abundances` property of an `verySimple` implementation of the `hotHalo` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `hothaloverysimpleabundancesjcbnzr`
Description: Indicate that the `abundances` property of an `verySimple` implementation of the `hotHalo` component class is inactive for differential equation solving.
Code lines: 14
Contained by: module `galacticus_nodes`

subroutine: `hothaloverysimpleabundancesrate`
Description: Accumulate to the rate of change of the `abundances` property of an `verySimple` implementation of the `hotHalo` component class.
Code lines: 38
Contained by: module `galacticus_nodes`

function: `hothaloverysimpleabundancesrateget`
Description: Get the rate of change of the `abundances` property of an `verySimple` implementation of the `hotHalo` component class.
Code lines: 26
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `hothaloverysimpleabundancesscale`
Description: Set the absolute scale of the `abundances` property of an `verySimple` implementation of the `hotHalo` component class.
Code lines: 15
Contained by: module `galacticus_nodes`

subroutine: `hothaloverysimpleabundancesset`
Description: Set the `abundances` property of an `verySimple` implementation of the `hotHalo` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `hothaloverysimpledelayedoutflowedabundancescount`
Description: Return a count of the number of scalar properties in the `outflowedAbundances` property of an `verySimpleDelayed` implementation of the `hotHalo` component class.
Code lines: 12
Contained by: module `galacticus_nodes`

function: `hothaloverysimpledelayedoutflowedabundancesget`
Description: Get the `outflowedAbundances` property of an `verySimpleDelayed` implementation of the `hotHalo` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `hothaloverysimpledelayedoutflowedabundancesjcbnzr`
Description: Indicate that the `outflowedAbundances` property of an `verySimpleDelayed` implementation of the `hotHalo` component class is inactive for differential equation solving.

Code lines: 14
Contained by: module galacticus_nodes

subroutine: hothaloverysimpledelayedoutflowedabundancesrate
Description: Accumulate to the rate of change of the outflowedAbundances property of an verySimpleDelayed implementation of the hotHalo component class.
Code lines: 38
Contained by: module galacticus_nodes

function: hothaloverysimpledelayedoutflowedabundancesrateget
Description: Get the rate of change of the outflowedAbundances property of an verySimpleDelayed implementation of the hotHalo component class.
Code lines: 26
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: hothaloverysimpledelayedoutflowedabundancesscale
Description: Set the absolute scale of the outflowedAbundances property of an verySimpleDelayed implementation of the hotHalo component class.
Code lines: 15
Contained by: module galacticus_nodes

subroutine: hothaloverysimpledelayedoutflowedabundancesset
Description: Set the outflowedAbundances property of an verySimpleDelayed implementation of the hotHalo component class.
Code lines: 10
Contained by: module galacticus_nodes

function: hothaloverysimpledelayedoutflowedmasscount
Description: Return a count of the number of scalar properties in the outflowedMass property of an verySimpleDelayed implementation of the hotHalo component class.
Code lines: 12
Contained by: module galacticus_nodes

function: hothaloverysimpledelayedoutflowedmassget
Description: Get the outflowedMass property of an verySimpleDelayed implementation of the hotHalo component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: hothaloverysimpledelayedoutflowedmassjcbnzr
Description: Indicate that the outflowedMass property of an verySimpleDelayed implementation of the hotHalo component class is inactive for differential equation solving.
Code lines: 12
Contained by: module galacticus_nodes

subroutine: hothaloverysimpledelayedoutflowedmassrate
Description: Accumulate to the rate of change of the outflowedMass property of an verySimpleDelayed implementation of the hotHalo component class.
Code lines: 31
Contained by: module galacticus_nodes

function: `hothaloverysimpledelayedoutflowedmassrateget`
Description: Get the rate of change of the `outflowedMass` property of an `verySimpleDelayed` implementation of the `hotHalo` component class.
Code lines: 23
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `hothaloverysimpledelayedoutflowedmassscale`
Description: Set the absolute scale of the `outflowedMass` property of an `verySimpleDelayed` implementation of the `hotHalo` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

subroutine: `hothaloverysimpledelayedoutflowedmassset`
Description: Set the `outflowedMass` property of an `verySimpleDelayed` implementation of the `hotHalo` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `hothaloverysimpledelayedoutputcount`
Description: Increment the count of properties to output for a `verySimpleDelayed` implementation of the `hotHalo` component.
Code lines: 16
Contained by: module `galacticus_nodes`

subroutine: `hothaloverysimpledelayedoutputnames`
Description: Return the names of properties to output for a `verySimpleDelayed` implementation of the `hotHalo` component.
Code lines: 28
Contained by: module `galacticus_nodes`

subroutine: `hothaloverysimpledelayedpostoutput`
Description: Perform post-output processing for a `verySimpleDelayed` implementation of the `hotHalo` component.
Code lines: 11
Contained by: module `galacticus_nodes`

function: `hothaloverysimplemasscount`
Description: Return a count of the number of scalar properties in the `mass` property of an `verySimple` implementation of the `hotHalo` component class.
Code lines: 12
Contained by: module `galacticus_nodes`

function: `hothaloverysimplemassget`
Description: Get the `mass` property of an `verySimple` implementation of the `hotHalo` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `hothaloverysimplemassjcbnzr`
Description: Indicate that the `mass` property of an `verySimple` implementation of the `hotHalo` component class is inactive for differential equation solving.
Code lines: 12
Contained by: module `galacticus_nodes`

subroutine: `hothaloverysimplemassrate`
Description: Accumulate to the rate of change of the `mass` property of an `verySimple` implementation of the `hotHalo` component class.
Code lines: 31
Contained by: module `galacticus_nodes`

function: `hothaloverysimplemassrateget`
Description: Get the rate of change of the `mass` property of an `verySimple` implementation of the `hotHalo` component class.
Code lines: 23
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `hothaloverysimplemassscale`
Description: Set the absolute scale of the `mass` property of an `verySimple` implementation of the `hotHalo` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

subroutine: `hothaloverysimplemassset`
Description: Set the `mass` property of an `verySimple` implementation of the `hotHalo` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `hothaloverysimpleouterradiusget`
Description: Get the value of the `outerRadius` property of the `verySimple` implementation of the `hotHalo` component using a deferred function.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `hothaloverysimpleouterradiusgetfunction`
Description: Set the function to be used for the `get` method of the `outerRadius` property of the `HotHaloVerySimple` component.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `hothaloverysimpleouterradiusgetisattached`
Description: Return true if the deferred function used to get the `outerRadius` property of the `HotHaloVerySimple` component class has been attached.
Code lines: 8
Contained by: module `galacticus_nodes`

subroutine: `hothaloverysimpleoutflowingabundancesrate`

Description: Accumulate the rate of change of the `outflowingAbundances` property of the `verySimple` implementation of the `hotHalo` component using a deferred function.
Code lines: 12
Contained by: module galacticus_nodes

subroutine: `hothaloverysimpleoutflowingabundancesratefunction`
Description: Set the function to be used for the `rate` method of the `outflowingAbundances` property of the `HotHaloVerySimple` component.
Code lines: 10
Contained by: module galacticus_nodes

function: `hothaloverysimpleoutflowingabundancesrateisattached`
Description: Return true if the deferred function used to rate the `outflowingAbundances` property of the `HotHaloVerySimple` component class has been attached.
Code lines: 8
Contained by: module galacticus_nodes

subroutine: `hothaloverysimpleoutflowingmassrate`
Description: Accumulate the rate of change of the `outflowingMass` property of the `verySimple` implementation of the `hotHalo` component using a deferred function.
Code lines: 12
Contained by: module galacticus_nodes

subroutine: `hothaloverysimpleoutflowingmassratefunction`
Description: Set the function to be used for the `rate` method of the `outflowingMass` property of the `HotHaloVerySimple` component.
Code lines: 10
Contained by: module galacticus_nodes

function: `hothaloverysimpleoutflowingmassrateisattached`
Description: Return true if the deferred function used to rate the `outflowingMass` property of the `HotHaloVerySimple` component class has been attached.
Code lines: 8
Contained by: module galacticus_nodes

subroutine: `hothaloverysimpleoutputcount`
Description: Increment the count of properties to output for a `verySimple` implementation of the `hotHalo` component.
Code lines: 18
Contained by: module galacticus_nodes

subroutine: `hothaloverysimpleoutputnames`
Description: Return the names of properties to output for a `verySimple` implementation of the `hotHalo` component.
Code lines: 36
Contained by: module galacticus_nodes

subroutine: `hothaloverysimplepostoutput`
Description: Perform post-output processing for a `verySimple` implementation of the `hotHalo` component.
Code lines: 10

Contained by: module galacticus_nodes

function: hothaloverysimpleunaccretedmasscount
Description: Return a count of the number of scalar properties in the unaccretedMass property of an verySimple implementation of the hotHalo component class.
Code lines: 12
Contained by: module galacticus_nodes

function: hothaloverysimpleunaccretedmassget
Description: Get the unaccretedMass property of an verySimple implementation of the hotHalo component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: hothaloverysimpleunaccretedmassjcbnzr
Description: Indicate that the unaccretedMass property of an verySimple implementation of the hotHalo component class is inactive for differential equation solving.
Code lines: 12
Contained by: module galacticus_nodes

subroutine: hothaloverysimpleunaccretedmassrate
Description: Accumulate to the rate of change of the unaccretedMass property of an verySimple implementation of the hotHalo component class.
Code lines: 31
Contained by: module galacticus_nodes

function: hothaloverysimpleunaccretedmassrateget
Description: Get the rate of change of the unaccretedMass property of an verySimple implementation of the hotHalo component class.
Code lines: 23
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: hothaloverysimpleunaccretedmassscale
Description: Set the absolute scale of the unaccretedMass property of an verySimple implementation of the hotHalo component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: hothaloverysimpleunaccretedmassset
Description: Set the unaccretedMass property of an verySimple implementation of the hotHalo component class.
Code lines: 10
Contained by: module galacticus_nodes

function: indicesbranchtip
Description: Returns the default value for the branchTip property for the indices component class.
Code lines: 13
Contained by: module galacticus_nodes

function: `indicesbranchtipattributematch`
Description: Return a text list of component implementations in the `indices` class that have the desired attributes for the `branchTip` property
Code lines: 30
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `indicesbranchtipisgettable`
Description: Returns true if the `branchTip` property is gettable for the `indices` component class.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `indicesbranchtiprateget`
Description: Returns a zero rate for the `branchTip` property for the `indices` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

subroutine: `indicesnulloutputcount`
Description: Increment the count of properties to output for a `null` implementation of the `indices` component.
Code lines: 22
Contained by: module `galacticus_nodes`

subroutine: `indicesnulloutputnames`
Description: Return the names of properties to output for a `null` implementation of the `indices` component.
Code lines: 39
Contained by: module `galacticus_nodes`

subroutine: `indicesoutput`
Description: Populate output buffers with properties to output for a `indices` component.
Code lines: 34
Contained by: module `galacticus_nodes`
Modules used: `multi_counters`

subroutine: `indicesoutputcount`
Description: Increment the count of properties to output for a generic `indices` component.
Code lines: 15
Contained by: module `galacticus_nodes`

subroutine: `indicesoutputnames`
Description: Establish the names of properties to output for a generic `indices` component.
Code lines: 20
Contained by: module `galacticus_nodes`

subroutine: `indicespostoutput`
Description: Perform post-output processing of a `indices` component.
Code lines: 14

Contained by: module galacticus_nodes

function: indicesstandardbranchtipget
Description: Get the branchTip property of an standard implementation of the indices component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: indicesstandardbranchtipset
Description: Set the branchTip property of an standard implementation of the indices component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: indicesstandardoutputcount
Description: Increment the count of properties to output for a standard implementation of the indices component.
Code lines: 21
Contained by: module galacticus_nodes

subroutine: indicesstandardoutputnames
Description: Return the names of properties to output for a standard implementation of the indices component.
Code lines: 37
Contained by: module galacticus_nodes

subroutine: interoutputcreatebyinterrupt
Description: Create the interOutput component of self via an interrupt.
Code lines: 10
Contained by: module galacticus_nodes

function: interoutputdiskstarformationrate
Description: Returns the default value for the diskStarFormationRate property for the interOutput component class.
Code lines: 17
Contained by: module galacticus_nodes

function: interoutputdiskstarformationrateattributematch
Description: Return a text list of component implementations in the interOutput class that have the desired attributes for the diskStarFormationRate property
Code lines: 28
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: interoutputdiskstarformationrateisgettable
Description: Returns true if the diskStarFormationRate property is gettable for the interOutput component class.
Code lines: 9
Contained by: module galacticus_nodes

subroutine: interoutputdiskstarformationraterate

Description: Accept a rate set for the `diskStarFormationRate` property of the `interOutput` component class. Trigger an interrupt to create the component.
Code lines: 21
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

function: `interoutputdiskstarformationraterateget`
Description: Returns a zero rate for the `diskStarFormationRate` property for the `interOutput` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

subroutine: `interoutputnulloutputcount`
Description: Increment the count of properties to output for a `null` implementation of the `interOutput` component.
Code lines: 22
Contained by: module `galacticus_nodes`

subroutine: `interoutputnulloutputnames`
Description: Return the names of properties to output for a `null` implementation of the `interOutput` component.
Code lines: 39
Contained by: module `galacticus_nodes`

subroutine: `interoutputoutput`
Description: Populate output buffers with properties to output for a `interOutput` component.
Code lines: 36
Contained by: module `galacticus_nodes`
Modules used: `multi_counters`

subroutine: `interoutputoutputcount`
Description: Increment the count of properties to output for a generic `interOutput` component.
Code lines: 15
Contained by: module `galacticus_nodes`

subroutine: `interoutputoutputnames`
Description: Establish the names of properties to output for a generic `interOutput` component.
Code lines: 20
Contained by: module `galacticus_nodes`

subroutine: `interoutputpostoutput`
Description: Perform post-output processing of a `interOutput` component.
Code lines: 14
Contained by: module `galacticus_nodes`

function: `interoutputspheroidstarformationrate`
Description: Returns the default value for the `spheroidStarFormationRate` property for the `interOutput` component class.
Code lines: 17
Contained by: module `galacticus_nodes`

function: `interoutputspheroidstarformationrateattributematch`
Description: Return a text list of component implementations in the `interOutput` class that have the desired attributes for the `spheroidStarFormationRate` property
Code lines: 28
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `interoutputspheroidstarformationrateisgettable`
Description: Returns true if the `spheroidStarFormationRate` property is gettable for the `interOutput` component class.
Code lines: 9
Contained by: module `galacticus_nodes`

subroutine: `interoutputspheroidstarformationraterate`
Description: Accept a rate set for the `spheroidStarFormationRate` property of the `interOutput` component class. Trigger an interrupt to create the component.
Code lines: 21
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

function: `interoutputspheroidstarformationraterateget`
Description: Returns a zero rate for the `spheroidStarFormationRate` property for the `interOutput` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `interoutputstandarddiskstarformationratecount`
Description: Return a count of the number of scalar properties in the `diskStarFormationRate` property of an `standard` implementation of the `interOutput` component class.
Code lines: 12
Contained by: module `galacticus_nodes`

function: `interoutputstandarddiskstarformationrateget`
Description: Get the `diskStarFormationRate` property of an `standard` implementation of the `interOutput` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `interoutputstandarddiskstarformationratejcbnzr`
Description: Indicate that the `diskStarFormationRate` property of an `standard` implementation of the `interOutput` component class is inactive for differential equation solving.
Code lines: 12
Contained by: module `galacticus_nodes`

subroutine: `interoutputstandarddiskstarformationraterate`
Description: Accumulate to the rate of change of the `diskStarFormationRate` property of an `standard` implementation of the `interOutput` component class.
Code lines: 31
Contained by: module `galacticus_nodes`

function: `interoutputstandarddiskstarformationraterateget`
Description: Get the rate of change of the `diskStarFormationRate` property of an `standard` implementation of the `interOutput` component class.
Code lines: 23
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `interoutputstandarddiskstarformationratescale`
Description: Set the absolute scale of the `diskStarFormationRate` property of an `standard` implementation of the `interOutput` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

subroutine: `interoutputstandarddiskstarformationrateset`
Description: Set the `diskStarFormationRate` property of an `standard` implementation of the `interOutput` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `interoutputstandardoutputcount`
Description: Increment the count of properties to output for a `standard` implementation of the `interOutput` component.
Code lines: 21
Contained by: module `galacticus_nodes`

subroutine: `interoutputstandardoutputnames`
Description: Return the names of properties to output for a `standard` implementation of the `interOutput` component.
Code lines: 43
Contained by: module `galacticus_nodes`

function: `interoutputstandardspheroidstarformationratecount`
Description: Return a count of the number of scalar properties in the `spheroidStarFormationRate` property of an `standard` implementation of the `interOutput` component class.
Code lines: 12
Contained by: module `galacticus_nodes`

function: `interoutputstandardspheroidstarformationrateget`
Description: Get the `spheroidStarFormationRate` property of an `standard` implementation of the `interOutput` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `interoutputstandardspheroidstarformationratejcbnzr`
Description: Indicate that the `spheroidStarFormationRate` property of an `standard` implementation of the `interOutput` component class is inactive for differential equation solving.
Code lines: 12
Contained by: module `galacticus_nodes`

subroutine: `interoutputstandardspheroidstarformationraterate`
Description: Accumulate to the rate of change of the `spheroidStarFormationRate` property of an `standard` implementation of the `interOutput` component class.
Code lines: 31
Contained by: module galacticus_nodes

function: `interoutputstandardspheroidstarformationraterateget`
Description: Get the rate of change of the `spheroidStarFormationRate` property of an `standard` implementation of the `interOutput` component class.
Code lines: 23
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: `interoutputstandardspheroidstarformationratescale`
Description: Set the absolute scale of the `spheroidStarFormationRate` property of an `standard` implementation of the `interOutput` component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: `interoutputstandardspheroidstarformationrateset`
Description: Set the `spheroidStarFormationRate` property of an `standard` implementation of the `interOutput` component class.
Code lines: 10
Contained by: module galacticus_nodes

function: `massflowstatisticscooledmass`
Description: Returns the default value for the `cooledMass` property for the `massFlowStatistics` component class.
Code lines: 13
Contained by: module galacticus_nodes

function: `massflowstatisticscooledmassattributematch`
Description: Return a text list of component implementations in the `massFlowStatistics` class that have the desired attributes for the `cooledMass` property
Code lines: 28
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: `massflowstatisticscooledmassisgettable`
Description: Returns true if the `cooledMass` property is gettable for the `massFlowStatistics` component class.
Code lines: 8
Contained by: module galacticus_nodes

function: `massflowstatisticscooledmassrateget`
Description: Returns a zero rate for the `cooledMass` property for the `massFlowStatistics` component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: `massflowstatisticsnulloutputcount`
Description: Increment the count of properties to output for a `null` implementation of the `massFlowStatistics` component.
Code lines: 22
Contained by: module `galacticus_nodes`

subroutine: `massflowstatisticsnulloutputnames`
Description: Return the names of properties to output for a `null` implementation of the `massFlowStatistics` component.
Code lines: 39
Contained by: module `galacticus_nodes`

subroutine: `massflowstatisticsoutput`
Description: Populate output buffers with properties to output for a `massFlowStatistics` component.
Code lines: 34
Contained by: module `galacticus_nodes`
Modules used: `multi_counters`

subroutine: `massflowstatisticsoutputcount`
Description: Increment the count of properties to output for a generic `massFlowStatistics` component.
Code lines: 15
Contained by: module `galacticus_nodes`

subroutine: `massflowstatisticsoutputnames`
Description: Establish the names of properties to output for a generic `massFlowStatistics` component.
Code lines: 20
Contained by: module `galacticus_nodes`

subroutine: `massflowstatisticspostoutput`
Description: Perform post-output processing of a `massFlowStatistics` component.
Code lines: 14
Contained by: module `galacticus_nodes`

function: `massflowstatisticsstandardcooledmasscount`
Description: Return a count of the number of scalar properties in the `cooledMass` property of an `standard` implementation of the `massFlowStatistics` component class.
Code lines: 12
Contained by: module `galacticus_nodes`

function: `massflowstatisticsstandardcooledmassget`
Description: Get the `cooledMass` property of an `standard` implementation of the `massFlowStatistics` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `massflowstatisticsstandardcooledmassjcbnzr`
Description: Indicate that the `cooledMass` property of an `standard` implementation of the `massFlowStatistics` component class is inactive for differential equation solving.
Code lines: 12

Contained by: module galacticus_nodes

subroutine: massflowstatisticsstandardcooledmassrate
Description: Accumulate to the rate of change of the cooledMass property of an standard implementation of the massFlowStatistics component class.
Code lines: 31
Contained by: module galacticus_nodes

function: massflowstatisticsstandardcooledmassrateget
Description: Get the rate of change of the cooledMass property of an standard implementation of the massFlowStatistics component class.
Code lines: 23
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: massflowstatisticsstandardcooledmassscale
Description: Set the absolute scale of the cooledMass property of an standard implementation of the massFlowStatistics component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: massflowstatisticsstandardcooledmassset
Description: Set the cooledMass property of an standard implementation of the massFlowStatistics component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: massflowstatisticsstandardoutputcount
Description: Increment the count of properties to output for a standard implementation of the massFlowStatistics component.
Code lines: 21
Contained by: module galacticus_nodes

subroutine: massflowstatisticsstandardoutputnames
Description: Return the names of properties to output for a standard implementation of the massFlowStatistics component.
Code lines: 37
Contained by: module galacticus_nodes

function: merger_tree_create_event
Description: Create a new event in a merger tree.
Code lines: 24
Contained by: module galacticus_nodes

subroutine: merger_tree_destroy
Description: Destroys the entire merger tree.
Code lines: 29
Contained by: module galacticus_nodes

function: merger_tree_earliest_time

Description: Return the earliest time in a merger tree.
Code lines: 26
Contained by: module galacticus_nodes

function: merger_tree_earliest_time_evolving
Description: Return the earliest time in a merger tree.
Code lines: 26
Contained by: module galacticus_nodes

function: merger_tree_latest_time
Description: Return the latest time in a merger tree.
Code lines: 20
Contained by: module galacticus_nodes

function: merger_tree_node_get
Description: Return a pointer to a node in tree given the index of the node.
Code lines: 25
Contained by: module galacticus_nodes

subroutine: merger_tree_remove_event
Description: Removed an event from self.
Code lines: 31
Contained by: module galacticus_nodes

function: merger_tree_size_of
Description: Return the size (in bytes) of a merger tree.
Code lines: 16
Contained by: module galacticus_nodes

function: merger_tree_walk_descend_to_progenitors
Description: Descend to the deepest progenitor (satellites and children) of self.
Code lines: 18
Contained by: module galacticus_nodes

interface: mergertree
Description: Interface to merger tree constructors.
Code lines: 5
Contained by: module galacticus_nodes

function: mergertreeconstructor
Description: Constructor for the merger tree class. Currently does nothing.
Code lines: 9
Contained by: module galacticus_nodes

type: mergertreelist
Description: A class used for building linked lists of merger trees.
Code lines: 6
Contained by: module galacticus_nodes

function: `mergingstatisticsgalaxymajormergertime`
Description: Returns the default value for the `galaxyMajorMergerTime` property for the `mergingStatistics` component class.
Code lines: 17
Contained by: module `galacticus_nodes`

function: `mergingstatisticsgalaxymajormergertimeattributematch`
Description: Return a text list of component implementations in the `mergingStatistics` class that have the desired attributes for the `galaxyMajorMergerTime` property
Code lines: 30
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `mergingstatisticsgalaxymajormergertimeisgettable`
Description: Returns true if the `galaxyMajorMergerTime` property is gettable for the `mergingStatistics` component class.
Code lines: 9
Contained by: module `galacticus_nodes`

function: `mergingstatisticsgalaxymajormergertimerateget`
Description: Returns a zero rate for the `galaxyMajorMergerTime` property for the `mergingStatistics` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `mergingstatisticsmajormajormergertimeget`
Description: Get the `majorMergerTime` property of an `major` implementation of the `mergingStatistics` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `mergingstatisticsmajormajormergertimeset`
Description: Set the `majorMergerTime` property of an `major` implementation of the `mergingStatistics` component class.
Code lines: 20
Contained by: module `galacticus_nodes`
Modules used: `memory_management`

function: `mergingstatisticsmajormergertime`
Description: Returns the default value for the `majorMergerTime` property for the `mergingStatistics` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `mergingstatisticsmajormergertimeattributematch`
Description: Return a text list of component implementations in the `mergingStatistics` class that have the desired attributes for the `majorMergerTime` property
Code lines: 30
Contained by: module `galacticus_nodes`

Modules used: iso_varying_string

function: mergingstatisticsmajormergertimeisgettable
Description: Returns true if the majorMergerTime property is gettable for the mergingStatistics component class.
Code lines: 8
Contained by: module galacticus_nodes

function: mergingstatisticsmajormergertimerateget
Description: Returns a zero rate for the majorMergerTime property for the mergingStatistics component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: mergingstatisticsmajoroutputcount
Description: Increment the count of properties to output for a major implementation of the mergingStatistics component.
Code lines: 22
Contained by: module galacticus_nodes

subroutine: mergingstatisticsmajoroutputnames
Description: Return the names of properties to output for a major implementation of the mergingStatistics component.
Code lines: 39
Contained by: module galacticus_nodes

function: mergingstatisticsmasswhenfirstisolated
Description: Returns the default value for the massWhenFirstIsolated property for the mergingStatistics component class.
Code lines: 13
Contained by: module galacticus_nodes

function: mergingstatisticsmasswhenfirstisolatedattributematch
Description: Return a text list of component implementations in the mergingStatistics class that have the desired attributes for the massWhenFirstIsolated property
Code lines: 30
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: mergingstatisticsmasswhenfirstisolatedisgettable
Description: Returns true if the massWhenFirstIsolated property is gettable for the mergingStatistics component class.
Code lines: 8
Contained by: module galacticus_nodes

function: mergingstatisticsmasswhenfirstisolatedrateget
Description: Returns a zero rate for the massWhenFirstIsolated property for the mergingStatistics component class.
Code lines: 13
Contained by: module galacticus_nodes

function: `mergingstatisticsnodeformationtime`
Description: Returns the default value for the `nodeFormationTime` property for the `mergingStatistics` component class.
Code lines: 17
Contained by: module `galacticus_nodes`

function: `mergingstatisticsnodeformationtimeattributematch`
Description: Return a text list of component implementations in the `mergingStatistics` class that have the desired attributes for the `nodeFormationTime` property
Code lines: 30
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `mergingstatisticsnodeformationtimeisgettable`
Description: Returns true if the `nodeFormationTime` property is gettable for the `mergingStatistics` component class.
Code lines: 9
Contained by: module `galacticus_nodes`

function: `mergingstatisticsnodeformationtimerateget`
Description: Returns a zero rate for the `nodeFormationTime` property for the `mergingStatistics` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `mergingstatisticsnodehierarchylevel`
Description: Returns the default value for the `nodeHierarchyLevel` property for the `mergingStatistics` component class.
Code lines: 17
Contained by: module `galacticus_nodes`

function: `mergingstatisticsnodehierarchylevelattributematch`
Description: Return a text list of component implementations in the `mergingStatistics` class that have the desired attributes for the `nodeHierarchyLevel` property
Code lines: 30
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `mergingstatisticsnodehierarchylevelisgettable`
Description: Returns true if the `nodeHierarchyLevel` property is gettable for the `mergingStatistics` component class.
Code lines: 9
Contained by: module `galacticus_nodes`

function: `mergingstatisticsnodehierarchylevelmaximum`
Description: Returns the default value for the `nodeHierarchyLevelMaximum` property for the `mergingStatistics` component class.
Code lines: 17
Contained by: module `galacticus_nodes`

function: `mergingstatisticsnodehierarchylevelmaximumattributematch`
Description: Return a text list of component implementations in the `mergingStatistics` class that have the desired attributes for the `nodeHierarchyLevelMaximum` property
Code lines: 30
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `mergingstatisticsnodehierarchylevelmaximumisgettable`
Description: Returns true if the `nodeHierarchyLevelMaximum` property is gettable for the `mergingStatistics` component class.
Code lines: 9
Contained by: module `galacticus_nodes`

function: `mergingstatisticsnodehierarchylevelmaximumrateget`
Description: Returns a zero rate for the `nodeHierarchyLevelMaximum` property for the `mergingStatistics` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `mergingstatisticsnodehierarchylevelrateget`
Description: Returns a zero rate for the `nodeHierarchyLevel` property for the `mergingStatistics` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `mergingstatisticsnodemajormergertime`
Description: Returns the default value for the `nodeMajorMergerTime` property for the `mergingStatistics` component class.
Code lines: 17
Contained by: module `galacticus_nodes`

function: `mergingstatisticsnodemajormergertimeattributematch`
Description: Return a text list of component implementations in the `mergingStatistics` class that have the desired attributes for the `nodeMajorMergerTime` property
Code lines: 30
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `mergingstatisticsnodemajormergertimeisgettable`
Description: Returns true if the `nodeMajorMergerTime` property is gettable for the `mergingStatistics` component class.
Code lines: 9
Contained by: module `galacticus_nodes`

function: `mergingstatisticsnodemajormergertimerateget`
Description: Returns a zero rate for the `nodeMajorMergerTime` property for the `mergingStatistics` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

subroutine: `mergingstatisticsnulloutputcount`
Description: Increment the count of properties to output for a `null` implementation of the `mergingStatistics` component.
Code lines: 22
Contained by: module `galacticus_nodes`

subroutine: `mergingstatisticsnulloutputnames`
Description: Return the names of properties to output for a `null` implementation of the `mergingStatistics` component.
Code lines: 39
Contained by: module `galacticus_nodes`

subroutine: `mergingstatisticsoutput`
Description: Populate output buffers with properties to output for a `mergingStatistics` component.
Code lines: 36
Contained by: module `galacticus_nodes`
Modules used: `multi_counters`

subroutine: `mergingstatisticsoutputcount`
Description: Increment the count of properties to output for a generic `mergingStatistics` component.
Code lines: 15
Contained by: module `galacticus_nodes`

subroutine: `mergingstatisticsoutputnames`
Description: Establish the names of properties to output for a generic `mergingStatistics` component.
Code lines: 20
Contained by: module `galacticus_nodes`

subroutine: `mergingstatisticspostoutput`
Description: Perform post-output processing of a `mergingStatistics` component.
Code lines: 14
Contained by: module `galacticus_nodes`

function: `mergingstatisticsrecentmajormergercount`
Description: Returns the default value for the `recentMajorMergerCount` property for the `mergingStatistics` component class.
Code lines: 20
Contained by: module `galacticus_nodes`
Modules used: `node_component_merging_statistics_-recent_data`

function: `mergingstatisticsrecentmajormergercountattributematch`
Description: Return a text list of component implementations in the `mergingStatistics` class that have the desired attributes for the `recentMajorMergerCount` property
Code lines: 30
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `mergingstatisticsrecentmajormergercountisgettable`

Description: Returns true if the `recentMajorMergerCount` property is gettable for the `mergingStatistics` component class.
Code lines: 9
Contained by: module galacticus_nodes

function: `mergingstatisticsrecentmajormergercountrateget`
Description: Returns a zero rate for the `recentMajorMergerCount` property for the `mergingStatistics` component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: `mergingstatisticsrecentoutputcount`
Description: Increment the count of properties to output for a `recent` implementation of the `mergingStatistics` component.
Code lines: 22
Contained by: module galacticus_nodes

subroutine: `mergingstatisticsrecentoutputnames`
Description: Return the names of properties to output for a `recent` implementation of the `mergingStatistics` component.
Code lines: 39
Contained by: module galacticus_nodes

function: `mergingstatisticsrecentrecentmajormergercountget`
Description: Get the `recentMajorMergerCount` property of an `recent` implementation of the `mergingStatistics` component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: `mergingstatisticsrecentrecentmajormergercountset`
Description: Set the `recentMajorMergerCount` property of an `recent` implementation of the `mergingStatistics` component class.
Code lines: 20
Contained by: module galacticus_nodes
Modules used: memory_management

function: `mergingstatisticsstandardgalaxymajormergertimeget`
Description: Get the `galaxyMajorMergerTime` property of an `standard` implementation of the `mergingStatistics` component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: `mergingstatisticsstandardgalaxymajormergertimeset`
Description: Set the `galaxyMajorMergerTime` property of an `standard` implementation of the `mergingStatistics` component class.
Code lines: 10
Contained by: module galacticus_nodes

function: `mergingstatisticsstandardmasswhenfirstisolatedget`
Description: Get the `massWhenFirstIsolated` property of an `standard` implementation of the `mergingStatistics` component class.

Code lines: 10
Contained by: module galacticus_nodes

subroutine: mergingstatisticsstandardmasswhenfirstisolatedset
Description: Set the massWhenFirstIsolated property of an standard implementation of the mergingStatistics component class.
Code lines: 10
Contained by: module galacticus_nodes

function: mergingstatisticsstandardnodeformationtimeget
Description: Get the nodeFormationTime property of an standard implementation of the mergingStatistics component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: mergingstatisticsstandardnodeformationtimeset
Description: Set the nodeFormationTime property of an standard implementation of the mergingStatistics component class.
Code lines: 10
Contained by: module galacticus_nodes

function: mergingstatisticsstandardnodehierarchylevelget
Description: Get the value of the nodeHierarchyLevel property of the standard implementation of the mergingStatistics component using a deferred function.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: mergingstatisticsstandardnodehierarchylevelgetfunction
Description: Set the function to be used for the get method of the nodeHierarchyLevel property of the MergingStatisticsStandard component.
Code lines: 10
Contained by: module galacticus_nodes

function: mergingstatisticsstandardnodehierarchylevelgetisattached
Description: Return true if the deferred function used to get the nodeHierarchyLevel property of the MergingStatisticsStandard component class has been attached.
Code lines: 8
Contained by: module galacticus_nodes

function: mergingstatisticsstandardnodehierarchylevelgetvalue
Description: Get the nodeHierarchyLevel property of an standard implementation of the mergingStatistics component class.
Code lines: 10
Contained by: module galacticus_nodes

function: mergingstatisticsstandardnodehierarchylevelmaximumget
Description: Get the value of the nodeHierarchyLevelMaximum property of the standard implementation of the mergingStatistics component using a deferred function.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: `mergingstatisticsstandardnodehierarchylevelmaximumgetfunction`
Description: Set the function to be used for the `get` method of the `nodeHierarchyLevelMaximum` property of the `MergingStatisticsStandard` component.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `mergingstatisticsstandardnodehierarchylevelmaximumgetisattached`
Description: Return true if the deferred function used to get the `nodeHierarchyLevelMaximum` property of the `MergingStatisticsStandard` component class has been attached.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `mergingstatisticsstandardnodehierarchylevelmaximumgetvalue`
Description: Get the `nodeHierarchyLevelMaximum` property of an `standard` implementation of the `mergingStatistics` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `mergingstatisticsstandardnodehierarchylevelmaximumset`
Description: Set the `nodeHierarchyLevelMaximum` property of an `standard` implementation of the `mergingStatistics` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `mergingstatisticsstandardnodehierarchylevelset`
Description: Set the `nodeHierarchyLevel` property of an `standard` implementation of the `mergingStatistics` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `mergingstatisticsstandardnodemajormergertimeget`
Description: Get the `nodeMajorMergerTime` property of an `standard` implementation of the `mergingStatistics` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `mergingstatisticsstandardnodemajormergertimeset`
Description: Set the `nodeMajorMergerTime` property of an `standard` implementation of the `mergingStatistics` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `mergingstatisticsstandardoutputcount`
Description: Increment the count of properties to output for a `standard` implementation of the `mergingStatistics` component.
Code lines: 20
Contained by: module `galacticus_nodes`

subroutine: `mergingstatisticsstandardoutputnames`

Description: Return the names of properties to output for a `standard` implementation of the `mergingStatistics` component.
Code lines: 53
Contained by: module `galacticus_nodes`

function: `nbodygenericaddintegerproperty`
Description: Call the deferred function for the `addIntegerProperty` method of the `nBody` component class if it has been set.
Code lines: 23
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `nbodygenericaddintegerpropertydeferredfunctionset`
Description: Set the function to be used for the `addIntegerProperty` method of the `generic` implementation of the `nBody` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `nbodygenericaddintegerpropertydfrrdfnctnisset`
Description: Return true if the deferred function for the `addIntegerProperty` method of the `generic` implementation of the `nBody` component class has been set.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `nbodygenericaddrealproperty`
Description: Call the deferred function for the `addRealProperty` method of the `nBody` component class if it has been set.
Code lines: 23
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `nbodygenericaddrealpropertydeferredfunctionset`
Description: Set the function to be used for the `addRealProperty` method of the `generic` implementation of the `nBody` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `nbodygenericaddrealpropertydfrrdfnctnisset`
Description: Return true if the deferred function for the `addRealProperty` method of the `generic` implementation of the `nBody` component class has been set.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `nbodygenericintegersget`
Description: Get the `integers` property of an `generic` implementation of the `nBody` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `nbodygenericintegersset`
Description: Set the `integers` property of an `generic` implementation of the `nBody` component class.

Code lines: 20
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nbodygenericoutputcount
Description: Increment the count of properties to output for a generic implementation of the nBody component.
Code lines: 22
Contained by: module galacticus_nodes

subroutine: nbodygenericoutputnames
Description: Return the names of properties to output for a generic implementation of the nBody component.
Code lines: 39
Contained by: module galacticus_nodes

function: nbodygenericrealsget
Description: Get the reals property of an generic implementation of the nBody component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: nbodygenericrealsset
Description: Set the reals property of an generic implementation of the nBody component class.
Code lines: 20
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nbodygenericsetintegerproperty
Description: Call the deferred function for the setIntegerProperty method of the nBody component class if it has been set.
Code lines: 22
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: nbodygenericsetintegerpropertydeferredfunctionset
Description: Set the function to be used for the setIntegerProperty method of the generic implementation of the nBody component class.
Code lines: 10
Contained by: module galacticus_nodes

function: nbodygenericsetintegerpropertydfrrdfnctnisset
Description: Return true if the deferred function for the setIntegerProperty method of the generic implementation of the nBody component class has been set.
Code lines: 8
Contained by: module galacticus_nodes

subroutine: nbodygenericsetrealproperty
Description: Call the deferred function for the setRealProperty method of the nBody component class if it has been set.
Code lines: 22

Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: nbodygenericsetrealpropertydeferredfunctionset
Description: Set the function to be used for the setRealProperty method of the generic implementation of the nBody component class.
Code lines: 10
Contained by: module galacticus_nodes

function: nbodygenericsetrealpropertydfrrdfnctnisset
Description: Return true if the deferred function for the setRealProperty method of the generic implementation of the nBody component class has been set.
Code lines: 8
Contained by: module galacticus_nodes

function: nbodyintegers
Description: Returns the default value for the integers property for the nBody component class.
Code lines: 13
Contained by: module galacticus_nodes

function: nbodyintegersattributematch
Description: Return a text list of component implementations in the nBody class that have the desired attributes for the integers property
Code lines: 30
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: nbodyintegersisgettable
Description: Returns true if the integers property is gettable for the nBody component class.
Code lines: 8
Contained by: module galacticus_nodes

function: nbodyintegersrateget
Description: Returns a zero rate for the integers property for the nBody component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: nbodynulloutputcount
Description: Increment the count of properties to output for a null implementation of the nBody component.
Code lines: 22
Contained by: module galacticus_nodes

subroutine: nbodynulloutputnames
Description: Return the names of properties to output for a null implementation of the nBody component.
Code lines: 39
Contained by: module galacticus_nodes

subroutine: nbodyoutput

Description: Populate output buffers with properties to output for a `nBody` component.
Code lines: 38
Contained by: module `galacticus_nodes`
Modules used: `multi_counters`

subroutine: `nbodyoutputcount`
Description: Increment the count of properties to output for a generic `nBody` component.
Code lines: 15
Contained by: module `galacticus_nodes`

subroutine: `nbodyoutputnames`
Description: Establish the names of properties to output for a generic `nBody` component.
Code lines: 20
Contained by: module `galacticus_nodes`

subroutine: `nbodypostoutput`
Description: Perform post-output processing of a `nBody` component.
Code lines: 14
Contained by: module `galacticus_nodes`

function: `nbodyreals`
Description: Returns the default value for the `reals` property for the `nBody` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `nbodyrealsattributematch`
Description: Return a text list of component implementations in the `nBody` class that have the desired attributes for the `reals` property
Code lines: 30
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `nbodyrealsisgettable`
Description: Returns true if the `reals` property is gettable for the `nBody` component class.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `nbodyrealsrateget`
Description: Returns a zero rate for the `reals` property for the `nBody` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `node_component_black_hole_simple_enclosed_mass`
Description: Computes the mass within a given radius for a central black hole. Black hole is treated as a point mass.
Code lines: 25
Contained by: module `galacticus_nodes`
Modules used: `galactic_structure_options`

function: `node_component_black_hole_simple_seed_mass`
Description: Return the seed mass for simple black holes.
Code lines: 43
Contained by: module `galacticus_nodes`
Modules used: `input_parameters`

function: `node_component_black_hole_standard_enclosed_mass`
Description: Computes the mass within a given radius for a central black hole. Black hole is treated as a point mass.
Code lines: 25
Contained by: module `galacticus_nodes`
Modules used: `galactic_structure_options`

function: `node_component_black_hole_standard_seed_mass`
Description: Return the seed mass for standard black holes.
Code lines: 43
Contained by: module `galacticus_nodes`
Modules used: `input_parameters`

function: `node_component_black_hole_standard_seed_spin`
Description: Return the seed spin for standard black holes.
Code lines: 15
Contained by: module `galacticus_nodes`
Modules used: `input_parameters`

function: `node_component_black_hole_standard_spin`
Description: Return the spin of a standard black hole.
Code lines: 12
Contained by: module `galacticus_nodes`

function: `node_component_density_null`
Description: A null implementation of the density in a component. Always returns zero.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `node_component_deserialize_null`
Description: Deserialize a generic tree node component.
Code lines: 9
Contained by: module `galacticus_nodes`

subroutine: `node_component_disk_standard_attach_pipes`
Description: Attach cooling pipes to the standard disk component.
Code lines: 21
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

function: `node_component_disk_standard_density`
Description: Computes the density at a given position for an standard disk.

Code lines: 49
Contained by: module galacticus_nodes
Modules used: coordinates, galactic_structure_options, node_component_disk_standard_data, numerical_constants_math, tables

function: node_component_disk_standard_enclosed_mass
Description: Computes the mass within a given radius for an standard disk.
Code lines: 49
Contained by: module galacticus_nodes
Modules used: galactic_structure_options, node_component_disk_standard_data, tables

function: node_component_disk_standard_half_mass_radius
Description: Return the half-mass radius of the standard disk.
Code lines: 20
Contained by: module galacticus_nodes
Modules used: galacticus_error, node_component_disk_standard_data

function: node_component_disk_standard_potential
Description: Compute the gravitational potential due to an standard disk.
Code lines: 41
Contained by: module galacticus_nodes
Modules used: bessel_functions, coordinates, galactic_structure_options, node_component_disk_standard_data, numerical_constants_physical, tables

function: node_component_disk_standard_rotation_curve
Description: Computes the rotation curve at a given radius for an standard disk.
Code lines: 32
Contained by: module galacticus_nodes
Modules used: galactic_structure_options, node_component_disk_standard_data, numerical_constants_physical, tables

function: node_component_disk_standard_rotation_curve_gradient
Description: Computes the rotation curve gradient for an standard disk.
Code lines: 35
Contained by: module galacticus_nodes
Modules used: galactic_structure_options, node_component_disk_standard_data, numerical_constants_physical, numerical_constants_prefixes, tables

function: node_component_disk_standard_surface_density
Description: Computes the surface density at a given position for an standard disk.
Code lines: 78
Contained by: module galacticus_nodes
Modules used: coordinates, galactic_structure_options, node_component_disk_standard_data, numerical_constants_math

tables

subroutine: `node_component_disk_very_simple_attach_pipe`
Description: Attach cooling pipes to the very simple disk component.
Code lines: 18
Contained by: module galacticus_nodes
Modules used: galacticus_error

function: `node_component_disk_very_simple_enclosed_mass`
Description: Computes the mass within a given radius for an very simple disk.
Code lines: 40
Contained by: module galacticus_nodes
Modules used: galactic_structure_options galacticus_error

function: `node_component_disk_very_simple_size_half_mass_radius`
Description: Return the half-mass radius of the very simple size disk.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: `node_component_dump_null`
Description: Dump a generic tree node component.
Code lines: 7
Contained by: module galacticus_nodes

subroutine: `node_component_dump_raw_null`
Description: Dump a generic tree node component in binary.
Code lines: 8
Contained by: module galacticus_nodes

subroutine: `node_component_dump_xml_null`
Description: Dump a generic tree node component to XML.
Code lines: 8
Contained by: module galacticus_nodes

subroutine: `node_component_dynamics_statistics_bars_record`
Description: Record the dynamical state.
Code lines: 47
Contained by: module galacticus_nodes
Modules used: memory_management

function: `node_component_enclosed_mass_null`
Description: A null implementation of the enclosed mass in a component. Always returns zero.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: `node_component_generic_destroy`
Description: Destroy a generic tree node component.
Code lines: 8
Contained by: module galacticus_nodes

function: `node_component_generic_type`
Description: Returns the name of a generic tree node component.
Code lines: 9
Contained by: module `galacticus_nodes`

function: `node_component_host_node`
Description: Return the host tree node of a tree node component.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `node_component_hot_halo_cold_mode_mass_total`
Description: Return the total active mass in the hot halo.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `node_component_hot_halo_outflow_tracking_mass_removal_rate`
Description: Handle instances where mass is removed from the the outflow tracking hot halo component class.
Code lines: 13
Contained by: module `galacticus_nodes`

subroutine: `node_component_hot_halo_standard_mass_removal_rate`
Description: Handle instances of mass removal from the standard hot halo component class. For the standard hot halo component type, we do nothing.
Code lines: 18
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

function: `node_component_hot_halo_standard_mass_total`
Description: Return the total active mass in the hot halo.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `node_component_null_double0_inout`
Description: A null `double` function for rank 0 `nodeComponent` arrays.
Code lines: 8
Contained by: module `galacticus_nodes`

subroutine: `node_component_null_void0_inout`
Description: A null `void` function for rank 0 `nodeComponent` arrays.
Code lines: 7
Contained by: module `galacticus_nodes`

subroutine: `node_component_ode_step_initialize_null`
Description: Initialize a generic tree node component for an ODE solver step.
Code lines: 7
Contained by: module `galacticus_nodes`

subroutine: `node_component_output_count_null`
Description: Dump a generic tree node component.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `node_component_output_names_null`
Description: Dump a generic tree node component.
Code lines: 12
Contained by: module `galacticus_nodes`

subroutine: `node_component_output_null`
Description: Dump a generic tree node component.
Code lines: 14
Contained by: module `galacticus_nodes`
Modules used: `multi_counters`

function: `node_component_potential_null`
Description: A null implementation of the gravitational potential in a component. Always returns zero.
Code lines: 11
Contained by: module `galacticus_nodes`

subroutine: `node_component_read_raw_null`
Description: Read a generic tree node component in binary.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `node_component_rotation_curve_gradient_null`
Description: A null implementation of the gradient of the rotation curve due to a component. Always returns zero.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `node_component_rotation_curve_null`
Description: A null implementation of the rotation curve due to a component. Always returns zero.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `node_component_satellite_orbiting_time_of_merging`
Description: Return the `timeOfMerging` property of the `satelliteOrbiting` component class.
Code lines: 19
Contained by: module `galacticus_nodes`

function: `node_component_satellite_preset_merge_time`
Description: Return the `mergeTime` property of the `satellitePreset` component class.
Code lines: 18
Contained by: module `galacticus_nodes`

subroutine: `node_component_satellite_preset_merge_time_set`

Description: Sets the `mergeTime` property of the `satellitePreset` component class.
Code lines: 15
Contained by: module galacticus_nodes

function: `node_component_satellite_standard_merge_time`
Description: Return the `mergeTime` property of the `satelliteStandard` component.
Code lines: 16
Contained by: module galacticus_nodes

function: `node_component_satellite_standard_time_of_merging`
Description: Return the `timeOfMerging` property of the `satelliteStandard` component class.
Code lines: 18
Contained by: module galacticus_nodes

subroutine: `node_component_satellite_standard_time_of_merging_set`
Description: Set the `timeOfMerging` property of the `satelliteStandard` component.
Code lines: 19
Contained by: module galacticus_nodes

function: `node_component_satellite_very_simple_merge_time`
Description: Return the `mergeTime` property of the `satelliteVerySimple` component.
Code lines: 16
Contained by: module galacticus_nodes

function: `node_component_satellite_very_simple_time_of_merging`
Description: Return the `timeOfMerging` property of the `satelliteVerySimple` component.
Code lines: 19
Contained by: module galacticus_nodes

subroutine: `node_component_serialization_offsets`
Description: Return the serialization count of a generic tree node component.
Code lines: 9
Contained by: module galacticus_nodes

function: `node_component_serialize_count_zero`
Description: Return the serialization count of a generic tree node component.
Code lines: 9
Contained by: module galacticus_nodes

subroutine: `node_component_serialize_null`
Description: Serialize a generic tree node component.
Code lines: 9
Contained by: module galacticus_nodes

function: `node_component_spheroid_standard_density`
Description: Computes the density at a given position for an standard spheroid.
Code lines: 55
Contained by: module galacticus_nodes
Modules used: coordinates galactic_structure_options

node_component_spheroid_standard_data numerical_constants_math

function: node_component_spheroid_standard_enclosed_mass
Description: Computes the mass within a given radius for an standard spheroid.
Code lines: 49
Contained by: module galacticus_nodes
Modules used: galactic_structure_options node_component_spheroid_standard_data

function: node_component_spheroid_standard_half_mass_radius
Description: Return the half-mass radius of the standard spheroid.
Code lines: 20
Contained by: module galacticus_nodes
Modules used: galacticus_error mass_distributions
node_component_spheroid_standard_data

function: node_component_spheroid_standard_potential
Description: Return the potential due to the standard spheroid.
Code lines: 37
Contained by: module galacticus_nodes
Modules used: coordinates galactic_structure_options
node_component_spheroid_standard_data numerical_constants_physical

function: node_component_spheroid_standard_rotation_curve
Description: Computes the rotation curve at a given radius for a standard spheroid.
Code lines: 24
Contained by: module galacticus_nodes
Modules used: galactic_structure_options numerical_constants_physical

function: node_component_spheroid_standard_rotation_curve_gradient
Description: Computes the rotation curve gradient for the standard spheroid.
Code lines: 29
Contained by: module galacticus_nodes
Modules used: galactic_structure_options numerical_constants_math
numerical_constants_physical numerical_constants_prefixes

function: node_component_spheroid_very_simple_enclosed_mass
Description: Computes the mass within a given radius for an very simple spheroid.
Code lines: 40
Contained by: module galacticus_nodes
Modules used: galactic_structure_options galacticus_error

function: node_component_spheroid_very_simple_half_mass_radius
Description: Return the half-mass radius of the very simple spheroid.
Code lines: 10
Contained by: module galacticus_nodes

function: node_component_surface_density_null
Description: A null implementation of the surface density in a component. Always returns zero.
Code lines: 10

Contained by: module galacticus_nodes

subroutine: nodeclasshierarchyfinalize
Description: Finalize the Galacticus node/component class hierarchy.
Code lines: 27
Contained by: module galacticus_nodes

subroutine: nodeclasshierarchyinitialize
Description: Initialize the Galacticus node/component class hierarchy.
Code lines: 853
Contained by: module galacticus_nodes
Modules used: input_parameters iso_varying_string memory_management

type: nodecomponent
Description: A class for components in nodes.
Code lines: 206
Contained by: module galacticus_nodes

type: nodecomponentagestatistics
Description: Type for the ageStatistics component class.
Code lines: 355
Contained by: module galacticus_nodes

subroutine: nodecomponentagestatisticsbuilder
Description: Build a generic ageStatistics component from a supplied XML definition.
Code lines: 19
Contained by: module galacticus_nodes
Modules used: fox_dom galacticus_error

subroutine: nodecomponentagestatisticscreate
Description: Create the ageStatistics component of self.
Code lines: 38
Contained by: module galacticus_nodes
Modules used: galacticus_display galacticus_error iso_varying_string string_handling

subroutine: nodecomponentagestatisticsdestroy
Description: Destroy the ageStatistics component of self
Code lines: 15
Contained by: module galacticus_nodes

subroutine: nodecomponentagestatisticsdumpascii
Description: Dump the content of a ageStatistics component.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string

subroutine: `nodecomponentagestatisticsfinalize`
Description: Finalize a generic `ageStatistics` component.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: `nodecomponentagestatisticsinitialize`
Description: Initialize a generic `ageStatistics` component.
Code lines: 15
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: `nodecomponentagestatisticsmove`
Description: Move instances of the `ageStatistics` component, from one node to another.
Code lines: 70
Contained by: module galacticus_nodes
Modules used: galacticus_error

type: `nodecomponentagestatisticsnull`
Description: Class for the null implementation of the ageStatistics component.
Code lines: 124
Contained by: module galacticus_nodes

subroutine: `nodecomponentagestatisticsnullbuilder`
Description: Build a `null` implementation of the `ageStatistics` component from a supplied XML definition.
Code lines: 18
Contained by: module galacticus_nodes
Modules used: fox_dom galacticus_error memory_management

subroutine: `nodecomponentagestatisticsnulldeserializeraw`
Description: Deserialize the contents of a null implementation of the ageStatistics component from raw (binary) file.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: `nodecomponentagestatisticsnulldeserializevalues`
Description: Deserialize evolvable properties of a null implementation of the ageStatistics component from array.
Code lines: 17
Contained by: module galacticus_nodes

subroutine: `nodecomponentagestatisticsnullfinalize`
Description: Finalize a `null` implementation of the `ageStatistics` component.
Code lines: 13
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: `nodecomponentagestatisticsnullinitialize`
Description: Initialize a `null` member of the `ageStatistics` component.
Code lines: 11
Contained by: module `galacticus_nodes`

function: `nodecomponentagestatisticsnullisactive`
Description: Return true if the null implementation of the ageStatistics component is the active choice.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `nodecomponentagestatisticsnullnamefromindex`
Description: Return the name of the property of given index for a `null` implementation of the `ageStatistics` component class.
Code lines: 21
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

subroutine: `nodecomponentagestatisticsnullserializeascii`
Description: Serialize the contents of a null implementation of the ageStatistics component to ASCII.
Code lines: 16
Contained by: module `galacticus_nodes`
Modules used: `galacticus_display` `iso_varying_string` `string_handling`

function: `nodecomponentagestatisticsnullserializecount`
Description: Return a count of the serialization of a null implementation of the ageStatistics component.
Code lines: 15
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentagestatisticsnullserializeoffsets`
Description: Compute offsets into serialization arrays for a `null` implementation of the `ageStatistics` component.
Code lines: 22
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentagestatisticsnullserializeraw`
Description: Serialize the contents of a null implementation of the ageStatistics component to raw (binary) file.
Code lines: 14
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentagestatisticsnullserializevalues`
Description: Serialize evolvable properties of a `null` implementation of the `ageStatistics` component to array.
Code lines: 17
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentagestatisticsnullserializexml`

Description: Serialize the contents of a null implementation of the ageStatistics component to XML.
Code lines: 14
Contained by: module galacticus_nodes

function: nodecomponentagestatisticsnullsizeof
Description: Return the size in bytes of a null implementation of the ageStatistics component.
Code lines: 9
Contained by: module galacticus_nodes

function: nodecomponentagestatisticsnulltype
Description: Returns the type name for the null implementation of the ageStatistics component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: nodecomponentagestatisticsremove
Description: Remove an instance of the ageStatistics component from a node.
Code lines: 42
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: nodecomponentagestatisticsserializeascii
Description: Serialize the content of a ageStatistics component to ASCII.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string

function: nodecomponentagestatisticssizeof
Description: Return the size in bytes of a nodeComponentAgeStatistics component.
Code lines: 9
Contained by: module galacticus_nodes

type: nodecomponentagestatisticsstandard
Description: Class for the standard implementation of the ageStatistics component.
Code lines: 333
Contained by: module galacticus_nodes

subroutine: nodecomponentagestatisticsstandardbuilder
Description: Build a standard implementation of the ageStatistics component from a supplied XML definition.
Code lines: 42
Contained by: module galacticus_nodes
Modules used: fox_dom galacticus_error memory_management

subroutine: nodecomponentagestatisticsstandarddeserializeraw
Description: Deserialize the contents of a standard implementation of the ageStatistics component from raw (binary) file.
Code lines: 15
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: `nodecomponentagestatisticsstandarddeserializevalues`
Description: Deserialize evolvable properties of a standard implementation of the ageStatistics component from array.
Code lines: 28
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentagestatisticsstandardfinalize`
Description: Finalize a `standard` implementation of the `ageStatistics` component.
Code lines: 13
Contained by: module `galacticus_nodes`
Modules used: `memory_management`

subroutine: `nodecomponentagestatisticsstandardinitialize`
Description: Initialize a `standard` member of the `ageStatistics` component.
Code lines: 12
Contained by: module `galacticus_nodes`

function: `nodecomponentagestatisticsstandardisactive`
Description: Return true if the standard implementation of the ageStatistics component is the active choice.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `nodecomponentagestatisticsstandardnamefromindex`
Description: Return the name of the property of given index for a `standard` implementation of the `ageStatistics` component class.
Code lines: 45
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

subroutine: `nodecomponentagestatisticsstandardserializeascii`
Description: Serialize the contents of a standard implementation of the ageStatistics component to ASCII.
Code lines: 28
Contained by: module `galacticus_nodes`
Modules used: `galacticus_display` `iso_varying_string` `string_handling`

function: `nodecomponentagestatisticsstandardserializecount`
Description: Return a count of the serialization of a standard implementation of the ageStatistics component.
Code lines: 46
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentagestatisticsstandardserializeoffsets`
Description: Compute offsets into serialization arrays for a `standard` implementation of the `ageStatistics` component.
Code lines: 70
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentagestatisticsstandardserializeraw`
Description: Serialize the contents of a standard implementation of the ageStatistics component to raw (binary) file.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: `nodecomponentagestatisticsstandardserializevalues`
Description: Serialize evolvable properties of a `standard` implementation of the `ageStatistics` component to array.
Code lines: 28
Contained by: module galacticus_nodes

subroutine: `nodecomponentagestatisticsstandardserializexml`
Description: Serialize the contents of a standard implementation of the ageStatistics component to XML.
Code lines: 18
Contained by: module galacticus_nodes

function: `nodecomponentagestatisticsstandardsizeof`
Description: Return the size in bytes of a standard implementation of the ageStatistics component.
Code lines: 9
Contained by: module galacticus_nodes

function: `nodecomponentagestatisticsstandardtype`
Description: Returns the type name for the standard implementation of the ageStatistics component class.
Code lines: 13
Contained by: module galacticus_nodes

function: `nodecomponentagestatisticstype`
Description: Returns the type name for the ageStatistics component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: `nodecomponentassign`
Description: Assign a node component to another node component.
Code lines: 459
Contained by: module galacticus_nodes

type: `nodecomponentbasic`
Description: Type for the `basic` component class.
Code lines: 635
Contained by: module galacticus_nodes

subroutine: `nodecomponentbasicbuilder`
Description: Build a generic `basic` component from a supplied XML definition.
Code lines: 19
Contained by: module galacticus_nodes
Modules used: `fox_dom` galacticus_error

subroutine: `nodecomponentbasiccreate`
Description: Create the `basic` component of `self`.
Code lines: 38
Contained by: module galacticus_nodes
Modules used: galacticus_display galacticus_error iso_varying_string string_handling

subroutine: `nodecomponentbasicdestroy`
Description: Destroy the `basic` component of `self`
Code lines: 15
Contained by: module galacticus_nodes

subroutine: `nodecomponentbasicdumpascii`
Description: Dump the content of a `basic` component.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string

type: `nodecomponentbasicextendedtracking`
Description: Class for the extendedTracking implementation of the basic component.
Code lines: 149
Contained by: module galacticus_nodes

subroutine: `nodecomponentbasicextendedtrackingbuilder`
Description: Build a `extendedTracking` implementation of the `basic` component from a supplied XML definition.
Code lines: 25
Contained by: module galacticus_nodes
Modules used: fox_dom galacticus_error memory_management

subroutine: `nodecomponentbasicextendedtrackingdeserializeraw`
Description: Deserialize the contents of a extendedTracking implementation of the basic component from raw (binary) file.
Code lines: 13
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: `nodecomponentbasicextendedtrackingdeserializevalues`
Description: Deserialize evolvable properties of a extendedTracking implementation of the basic component from array.
Code lines: 17
Contained by: module galacticus_nodes

subroutine: `nodecomponentbasicextendedtrackingfinalize`
Description: Finalize a `extendedTracking` implementation of the `basic` component.
Code lines: 13
Contained by: module galacticus_nodes

Modules used: memory_management

subroutine: nodecomponentbasicextendedtrackinginitialize
Description: Initialize a extendedTracking member of the basic component.
Code lines: 10
Contained by: module galacticus_nodes

function: nodecomponentbasicextendedtrackingisactive
Description: Return true if the extendedTracking implementation of the basic component is the active choice.
Code lines: 8
Contained by: module galacticus_nodes

function: nodecomponentbasicextendedtrackingnamefromindex
Description: Return the name of the property of given index for a extendedTracking implementation of the basic component class.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: iso_varying_string

subroutine: nodecomponentbasicextendedtrackingserializeascii
Description: Serialize the contents of a extendedTracking implementation of the basic component to ASCII.
Code lines: 20
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string string_handling

function: nodecomponentbasicextendedtrackingserializecount
Description: Return a count of the serialization of a extendedTracking implementation of the basic component.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: nodecomponentbasicextendedtrackingserializeoffsets
Description: Compute offsets into serialization arrays for a extendedTracking implementation of the basic component.
Code lines: 15
Contained by: module galacticus_nodes

subroutine: nodecomponentbasicextendedtrackingserializeraw
Description: Serialize the contents of a extendedTracking implementation of the basic component to raw (binary) file.
Code lines: 11
Contained by: module galacticus_nodes

subroutine: nodecomponentbasicextendedtrackingserializevalues
Description: Serialize evolvable properties of a extendedTracking implementation of the basic component to array.
Code lines: 17
Contained by: module galacticus_nodes

subroutine: `nodecomponentbasicextendedtrackingserializexml`
Description: Serialize the contents of a extendedTracking implementation of the basic component to XML.
Code lines: 16
Contained by: module galacticus_nodes

function: `nodecomponentbasicextendedtrackingsizeof`
Description: Return the size in bytes of a extendedTracking implementation of the basic component.
Code lines: 9
Contained by: module galacticus_nodes

function: `nodecomponentbasicextendedtrackingtype`
Description: Returns the type name for the extendedTracking implementation of the basic component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: `nodecomponentbasicfinalize`
Description: Finalize a generic `basic` component.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: `nodecomponentbasicinitialize`
Description: Initialize a generic `basic` component.
Code lines: 15
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: `nodecomponentbasicmove`
Description: Move instances of the `basic` component, from one node to another.
Code lines: 126
Contained by: module galacticus_nodes
Modules used: galacticus_error

type: `nodecomponentbasicnonevolving`
Description: Class for the nonEvolving implementation of the basic component.
Code lines: 205
Contained by: module galacticus_nodes

subroutine: `nodecomponentbasicnonevolvingbuilder`
Description: Build a `nonEvolving` implementation of the `basic` component from a supplied XML definition.
Code lines: 36
Contained by: module galacticus_nodes
Modules used: fox_dom galacticus_error memory_management

subroutine: `nodecomponentbasicnonevolvingdeserializeraw`
Description: Deserialize the contents of a nonEvolving implementation of the basic component from raw (binary) file.

Code lines: 14
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponentbasicnonevolvingdeserializevalues
Description: Deserialize evolvable properties of a nonEvolving implementation of the basic component from array.
Code lines: 16
Contained by: module galacticus_nodes

subroutine: nodecomponentbasicnonevolvingfinalize
Description: Finalize a nonEvolving implementation of the basic component.
Code lines: 13
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponentbasicnonevolvinginitialize
Description: Initialize a nonEvolving member of the basic component.
Code lines: 11
Contained by: module galacticus_nodes

function: nodecomponentbasicnonevolvingisactive
Description: Return true if the nonEvolving implementation of the basic component is the active choice.
Code lines: 8
Contained by: module galacticus_nodes

function: nodecomponentbasicnonevolvingnamefromindex
Description: Return the name of the property of given index for a nonEvolving implementation of the basic component class.
Code lines: 24
Contained by: module galacticus_nodes
Modules used: iso_varying_string

subroutine: nodecomponentbasicnonevolvingserializeascii
Description: Serialize the contents of a nonEvolving implementation of the basic component to ASCII.
Code lines: 25
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string string_handling

function: nodecomponentbasicnonevolvingserializecount
Description: Return a count of the serialization of a nonEvolving implementation of the basic component.
Code lines: 22
Contained by: module galacticus_nodes

subroutine: nodecomponentbasicnonevolvingserializeoffsets
Description: Compute offsets into serialization arrays for a nonEvolving implementation of the basic component.
Code lines: 28
Contained by: module galacticus_nodes

subroutine: `nodecomponentbasicnonevolvingserializeraw`
Description: Serialize the contents of a nonEvolving implementation of the basic component to raw (binary) file.
Code lines: 12
Contained by: module galacticus_nodes

subroutine: `nodecomponentbasicnonevolvingserializevalues`
Description: Serialize evolvable properties of a `nonEvolving` implementation of the `basic` component to array.
Code lines: 16
Contained by: module galacticus_nodes

subroutine: `nodecomponentbasicnonevolvingserializexml`
Description: Serialize the contents of a nonEvolving implementation of the basic component to XML.
Code lines: 17
Contained by: module galacticus_nodes

function: `nodecomponentbasicnonevolvingsizeof`
Description: Return the size in bytes of a nonEvolving implementation of the basic component.
Code lines: 9
Contained by: module galacticus_nodes

function: `nodecomponentbasicnonevolvingtype`
Description: Returns the type name for the nonEvolving implementation of the basic component class.
Code lines: 13
Contained by: module galacticus_nodes

type: `nodecomponentbasicnull`
Description: Class for the null implementation of the basic component.
Code lines: 124
Contained by: module galacticus_nodes

subroutine: `nodecomponentbasicnullbuilder`
Description: Build a `null` implementation of the `basic` component from a supplied XML definition.
Code lines: 18
Contained by: module galacticus_nodes
Modules used: `fox_dom` galacticus_error memory_management

subroutine: `nodecomponentbasicnulldeserializeraw`
Description: Deserialize the contents of a null implementation of the basic component from raw (binary) file.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: `nodecomponentbasicnulldeserializevalues`
Description: Deserialize evolvable properties of a null implementation of the basic component from array.

Code lines: 17
Contained by: module galacticus_nodes

subroutine: nodecomponentbasicnullfinalize
Description: Finalize a null implementation of the basic component.
Code lines: 13
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponentbasicnullinitialize
Description: Initialize a null member of the basic component.
Code lines: 11
Contained by: module galacticus_nodes

function: nodecomponentbasicnullisactive
Description: Return true if the null implementation of the basic component is the active choice.
Code lines: 8
Contained by: module galacticus_nodes

function: nodecomponentbasicnullnamefromindex
Description: Return the name of the property of given index for a null implementation of the basic component class.
Code lines: 21
Contained by: module galacticus_nodes
Modules used: iso_varying_string

subroutine: nodecomponentbasicnullserializeascii
Description: Serialize the contents of a null implementation of the basic component to ASCII.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string string_handling

function: nodecomponentbasicnullserializecount
Description: Return a count of the serialization of a null implementation of the basic component.
Code lines: 15
Contained by: module galacticus_nodes

subroutine: nodecomponentbasicnullserializeoffsets
Description: Compute offsets into serialization arrays for a null implementation of the basic component.
Code lines: 22
Contained by: module galacticus_nodes

subroutine: nodecomponentbasicnullserializeraw
Description: Serialize the contents of a null implementation of the basic component to raw (binary) file.
Code lines: 14
Contained by: module galacticus_nodes

subroutine: nodecomponentbasicnullserializevalues

Description: Serialize evolvable properties of a `null` implementation of the `basic` component to array.
Code lines: 17
Contained by: module galacticus_nodes

subroutine: `nodecomponentbasicnullserializexml`
Description: Serialize the contents of a null implementation of the basic component to XML.
Code lines: 14
Contained by: module galacticus_nodes

function: `nodecomponentbasicnullsizeof`
Description: Return the size in bytes of a null implementation of the basic component.
Code lines: 9
Contained by: module galacticus_nodes

function: `nodecomponentbasicnulltype`
Description: Returns the type name for the null implementation of the basic component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: `nodecomponentbasicremove`
Description: Remove an instance of the `basic` component from a node.
Code lines: 74
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: `nodecomponentbasicserializeascii`
Description: Serialize the content of a `basic` component to ASCII.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string

function: `nodecomponentbasicsizeof`
Description: Return the size in bytes of a nodeComponentBasic component.
Code lines: 9
Contained by: module galacticus_nodes

type: `nodecomponentbasicstandard`
Description: Class for the standard implementation of the basic component.
Code lines: 274
Contained by: module galacticus_nodes

subroutine: `nodecomponentbasicstandardbuilder`
Description: Build a `standard` implementation of the `basic` component from a supplied XML definition.
Code lines: 48
Contained by: module galacticus_nodes
Modules used: fox_dom galacticus_error memory_management

subroutine: `nodecomponentbasicstandarddeserializeraw`
Description: Deserialize the contents of a standard implementation of the basic component from raw (binary) file.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: `nodecomponentbasicstandarddeserializevalues`
Description: Deserialize evolvable properties of a standard implementation of the basic component from array.
Code lines: 20
Contained by: module galacticus_nodes

type: `nodecomponentbasicstandardextended`
Description: Class for the standardExtended implementation of the basic component.
Code lines: 312
Contained by: module galacticus_nodes

subroutine: `nodecomponentbasicstandardextendedbuilder`
Description: Build a `standardExtended` implementation of the `basic` component from a supplied XML definition.
Code lines: 43
Contained by: module galacticus_nodes
Modules used: fox_dom galacticus_error memory_management

subroutine: `nodecomponentbasicstandardextendeddeserializeraw`
Description: Deserialize the contents of a standardExtended implementation of the basic component from raw (binary) file.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: `nodecomponentbasicstandardextendeddeserializevalues`
Description: Deserialize evolvable properties of a standardExtended implementation of the basic component from array.
Code lines: 25
Contained by: module galacticus_nodes

subroutine: `nodecomponentbasicstandardextendedfinalize`
Description: Finalize a `standardExtended` implementation of the `basic` component.
Code lines: 13
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: `nodecomponentbasicstandardextendedinitialize`
Description: Initialize a `standardExtended` member of the `basic` component.
Code lines: 13

Contained by: module galacticus_nodes

function: nodecomponentbasicstandardextendedisactive
Description: Return true if the standardExtended implementation of the basic component is the active choice.
Code lines: 8
Contained by: module galacticus_nodes

function: nodecomponentbasicstandardextendednamefromindex
Description: Return the name of the property of given index for a standardExtended implementation of the basic component class.
Code lines: 30
Contained by: module galacticus_nodes
Modules used: iso_varying_string

subroutine: nodecomponentbasicstandardextendedserializeascii
Description: Serialize the contents of a standardExtended implementation of the basic component to ASCII.
Code lines: 29
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string string_handling

function: nodecomponentbasicstandardextendedserializecount
Description: Return a count of the serialization of a standardExtended implementation of the basic component.
Code lines: 27
Contained by: module galacticus_nodes

subroutine: nodecomponentbasicstandardextendedserializeoffsets
Description: Compute offsets into serialization arrays for a standardExtended implementation of the basic component.
Code lines: 40
Contained by: module galacticus_nodes

subroutine: nodecomponentbasicstandardextendedserializeraw
Description: Serialize the contents of a standardExtended implementation of the basic component to raw (binary) file.
Code lines: 14
Contained by: module galacticus_nodes

subroutine: nodecomponentbasicstandardextendedserializevalues
Description: Serialize evolvable properties of a standardExtended implementation of the basic component to array.
Code lines: 25
Contained by: module galacticus_nodes

subroutine: nodecomponentbasicstandardextendedserializexml
Description: Serialize the contents of a standardExtended implementation of the basic component to XML.
Code lines: 19

Contained by: module galacticus_nodes

function: nodecomponentbasicstandardextendedsizeof
Description: Return the size in bytes of a standardExtended implementation of the basic component.
Code lines: 9
Contained by: module galacticus_nodes

function: nodecomponentbasicstandardextendedtype
Description: Returns the type name for the standardExtended implementation of the basic component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: nodecomponentbasicstandardfinalize
Description: Finalize a standard implementation of the basic component.
Code lines: 13
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponentbasicstandardinitialize
Description: Initialize a standard member of the basic component.
Code lines: 13
Contained by: module galacticus_nodes

function: nodecomponentbasicstandardisactive
Description: Return true if the standard implementation of the basic component is the active choice.
Code lines: 8
Contained by: module galacticus_nodes

function: nodecomponentbasicstandardnamefromindex
Description: Return the name of the property of given index for a standard implementation of the basic component class.
Code lines: 31
Contained by: module galacticus_nodes
Modules used: iso_varying_string

subroutine: nodecomponentbasicstandardserializeascii
Description: Serialize the contents of a standard implementation of the basic component to ASCII.
Code lines: 31
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string string_handling

function: nodecomponentbasicstandardserializecount
Description: Return a count of the serialization of a standard implementation of the basic component.
Code lines: 30
Contained by: module galacticus_nodes

subroutine: nodecomponentbasicstandardserializeoffsets

Description: Compute offsets into serialization arrays for a `standard` implementation of the `basic` component.
Code lines: 42
Contained by: module galacticus_nodes

subroutine: `nodecomponentbasicstandardserializeraw`
Description: Serialize the contents of a standard implementation of the basic component to raw (binary) file.
Code lines: 14
Contained by: module galacticus_nodes

subroutine: `nodecomponentbasicstandardserializevalues`
Description: Serialize evolvable properties of a `standard` implementation of the `basic` component to array.
Code lines: 20
Contained by: module galacticus_nodes

subroutine: `nodecomponentbasicstandardserializexml`
Description: Serialize the contents of a standard implementation of the basic component to XML.
Code lines: 19
Contained by: module galacticus_nodes

function: `nodecomponentbasicstandardsizeof`
Description: Return the size in bytes of a standard implementation of the basic component.
Code lines: 9
Contained by: module galacticus_nodes

type: `nodecomponentbasicstandardtracking`
Description: Class for the standardTracking implementation of the basic component.
Code lines: 149
Contained by: module galacticus_nodes

subroutine: `nodecomponentbasicstandardtrackingbuilder`
Description: Build a `standardTracking` implementation of the `basic` component from a supplied XML definition.
Code lines: 25
Contained by: module galacticus_nodes
Modules used: `fox_dom` galacticus_error memory_management

subroutine: `nodecomponentbasicstandardtrackingdeserializeraw`
Description: Deserialize the contents of a standardTracking implementation of the basic component from raw (binary) file.
Code lines: 13
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: `nodecomponentbasicstandardtrackingdeserializevalues`
Description: Deserialize evolvable properties of a standardTracking implementation of the basic component from array.
Code lines: 17

Contained by: module galacticus_nodes

subroutine: nodecomponentbasicstandardtrackingfinalize
Description: Finalize a standardTracking implementation of the basic component.
Code lines: 13
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponentbasicstandardtrackinginitialize
Description: Initialize a standardTracking member of the basic component.
Code lines: 10
Contained by: module galacticus_nodes

function: nodecomponentbasicstandardtrackingisactive
Description: Return true if the standardTracking implementation of the basic component is the active choice.
Code lines: 8
Contained by: module galacticus_nodes

function: nodecomponentbasicstandardtrackingnamefromindex
Description: Return the name of the property of given index for a standardTracking implementation of the basic component class.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: iso_varying_string

subroutine: nodecomponentbasicstandardtrackingserializeascii
Description: Serialize the contents of a standardTracking implementation of the basic component to ASCII.
Code lines: 20
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string string_handling

function: nodecomponentbasicstandardtrackingserializecount
Description: Return a count of the serialization of a standardTracking implementation of the basic component.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: nodecomponentbasicstandardtrackingserializeoffsets
Description: Compute offsets into serialization arrays for a standardTracking implementation of the basic component.
Code lines: 15
Contained by: module galacticus_nodes

subroutine: nodecomponentbasicstandardtrackingserializeraw
Description: Serialize the contents of a standardTracking implementation of the basic component to raw (binary) file.
Code lines: 11
Contained by: module galacticus_nodes

subroutine: `nodecomponentbasicstandardtrackingserializevalues`
Description: Serialize evolvable properties of a `standardTracking` implementation of the `basic` component to array.
Code lines: 17
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentbasicstandardtrackingserializexml`
Description: Serialize the contents of a standardTracking implementation of the basic component to XML.
Code lines: 16
Contained by: module `galacticus_nodes`

function: `nodecomponentbasicstandardtrackingsizeof`
Description: Return the size in bytes of a standardTracking implementation of the basic component.
Code lines: 9
Contained by: module `galacticus_nodes`

function: `nodecomponentbasicstandardtrackingtype`
Description: Returns the type name for the standardTracking implementation of the basic component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `nodecomponentbasicstandardtype`
Description: Returns the type name for the standard implementation of the basic component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `nodecomponentbasictype`
Description: Returns the type name for the basic component class.
Code lines: 13
Contained by: module `galacticus_nodes`

type: `nodecomponentblackhole`
Description: Type for the `blackHole` component class.
Code lines: 488
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentblackholebuilder`
Description: Build a generic `blackHole` component from a supplied XML definition.
Code lines: 19
Contained by: module `galacticus_nodes`
Modules used: `fox_dom` `galacticus_error`

subroutine: `nodecomponentblackholecreate`
Description: Create the `blackHole` component of `self`.
Code lines: 38
Contained by: module `galacticus_nodes`
Modules used: `galacticus_display` `galacticus_error`

iso_varying_string string_handling

subroutine: `nodecomponentblackholedestroy`
Description: Destroy the `blackHole` component of `self`
Code lines: 15
Contained by: module galacticus_nodes

subroutine: `nodecomponentblackholedumpascii`
Description: Dump the content of a `blackHole` component.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string

subroutine: `nodecomponentblackholefinalize`
Description: Finalize a generic `blackHole` component.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: `nodecomponentblackholeinitialize`
Description: Initialize a generic `blackHole` component.
Code lines: 15
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: `nodecomponentblackholemove`
Description: Move instances of the `blackHole` component, from one node to another.
Code lines: 98
Contained by: module galacticus_nodes
Modules used: galacticus_error

type: `nodecomponentblackholenoncentral`
Description: Class for the nonCentral implementation of the blackHole component.
Code lines: 182
Contained by: module galacticus_nodes

subroutine: `nodecomponentblackholenoncentralbuilder`
Description: Build a `nonCentral` implementation of the `blackHole` component from a supplied XML definition.
Code lines: 25
Contained by: module galacticus_nodes
Modules used: fox_dom galacticus_error memory_management

subroutine: `nodecomponentblackholenoncentraldeserializeraw`
Description: Deserialize the contents of a nonCentral implementation of the blackHole component from raw (binary) file.
Code lines: 13
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: `nodecomponentblackholenoncentraldeserializevalues`
Description: Deserialize evolvable properties of a nonCentral implementation of the blackHole component from array.
Code lines: 21
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentblackholenoncentralfinalize`
Description: Finalize a `nonCentral` implementation of the `blackHole` component.
Code lines: 13
Contained by: module `galacticus_nodes`
Modules used: `memory_management`

subroutine: `nodecomponentblackholenoncentralinitialize`
Description: Initialize a `nonCentral` member of the `blackHole` component.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `nodecomponentblackholenoncentralisactive`
Description: Return true if the nonCentral implementation of the blackHole component is the active choice.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `nodecomponentblackholenoncentralnamefromindex`
Description: Return the name of the property of given index for a `nonCentral` implementation of the `blackHole` component class.
Code lines: 23
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

subroutine: `nodecomponentblackholenoncentralserializeascii`
Description: Serialize the contents of a nonCentral implementation of the blackHole component to ASCII.
Code lines: 20
Contained by: module `galacticus_nodes`
Modules used: `galacticus_display` `iso_varying_string` `string_handling`

function: `nodecomponentblackholenoncentralserializecount`
Description: Return a count of the serialization of a nonCentral implementation of the blackHole component.
Code lines: 19
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentblackholenoncentralserializeoffsets`
Description: Compute offsets into serialization arrays for a `nonCentral` implementation of the `blackHole` component.
Code lines: 26
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentblackholenoncentralserializeraw`
Description: Serialize the contents of a nonCentral implementation of the blackHole component to raw (binary) file.
Code lines: 11
Contained by: module galacticus_nodes

subroutine: `nodecomponentblackholenoncentralserializevalues`
Description: Serialize evolvable properties of a `nonCentral` implementation of the `blackHole` component to array.
Code lines: 21
Contained by: module galacticus_nodes

subroutine: `nodecomponentblackholenoncentralserializexml`
Description: Serialize the contents of a nonCentral implementation of the blackHole component to XML.
Code lines: 16
Contained by: module galacticus_nodes

function: `nodecomponentblackholenoncentralsizeof`
Description: Return the size in bytes of a nonCentral implementation of the blackHole component.
Code lines: 9
Contained by: module galacticus_nodes

function: `nodecomponentblackholenoncentraltype`
Description: Returns the type name for the nonCentral implementation of the blackHole component class.
Code lines: 13
Contained by: module galacticus_nodes

type: `nodecomponentblackholenull`
Description: Class for the null implementation of the blackHole component.
Code lines: 124
Contained by: module galacticus_nodes

subroutine: `nodecomponentblackholenullbuilder`
Description: Build a `null` implementation of the `blackHole` component from a supplied XML definition.
Code lines: 18
Contained by: module galacticus_nodes
Modules used: `fox_dom` galacticus_error memory_management

subroutine: `nodecomponentblackholenulldeserializeraw`
Description: Deserialize the contents of a null implementation of the blackHole component from raw (binary) file.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: `nodecomponentblackholenulldeserializevalues`
Description: Deserialize evolvable properties of a null implementation of the blackHole component from array.

Code lines: 17
Contained by: module galacticus_nodes

subroutine: nodecomponentblackholenullfinalize
Description: Finalize a null implementation of the blackHole component.
Code lines: 13
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponentblackholenullinitialize
Description: Initialize a null member of the blackHole component.
Code lines: 11
Contained by: module galacticus_nodes

function: nodecomponentblackholenullisactive
Description: Return true if the null implementation of the blackHole component is the active choice.
Code lines: 8
Contained by: module galacticus_nodes

function: nodecomponentblackholenullnamefromindex
Description: Return the name of the property of given index for a null implementation of the blackHole component class.
Code lines: 21
Contained by: module galacticus_nodes
Modules used: iso_varying_string

subroutine: nodecomponentblackholenullserializeascii
Description: Serialize the contents of a null implementation of the blackHole component to ASCII.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string string_handling

function: nodecomponentblackholenullserializecount
Description: Return a count of the serialization of a null implementation of the blackHole component.
Code lines: 15
Contained by: module galacticus_nodes

subroutine: nodecomponentblackholenullserializeoffsets
Description: Compute offsets into serialization arrays for a null implementation of the blackHole component.
Code lines: 22
Contained by: module galacticus_nodes

subroutine: nodecomponentblackholenullserializeraw
Description: Serialize the contents of a null implementation of the blackHole component to raw (binary) file.
Code lines: 14
Contained by: module galacticus_nodes

subroutine: `nodecomponentblackholenullserializevalues`
Description: Serialize evolvable properties of a `null` implementation of the `blackHole` component to array.
Code lines: 17
Contained by: module galacticus_nodes

subroutine: `nodecomponentblackholenullserializexml`
Description: Serialize the contents of a null implementation of the blackHole component to XML.
Code lines: 14
Contained by: module galacticus_nodes

function: `nodecomponentblackholenullsizeof`
Description: Return the size in bytes of a null implementation of the blackHole component.
Code lines: 9
Contained by: module galacticus_nodes

function: `nodecomponentblackholenulltype`
Description: Returns the type name for the null implementation of the blackHole component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: `nodecomponentblackholeremove`
Description: Remove an instance of the `blackHole` component from a node.
Code lines: 58
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: `nodecomponentblackholeserializeascii`
Description: Serialize the content of a `blackHole` component to ASCII.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string

type: `nodecomponentblackholesimple`
Description: Class for the simple implementation of the blackHole component.
Code lines: 181
Contained by: module galacticus_nodes

subroutine: `nodecomponentblackholesimplebuilder`
Description: Build a `simple` implementation of the `blackHole` component from a supplied XML definition.
Code lines: 24
Contained by: module galacticus_nodes
Modules used: fox_dom galacticus_error memory_management

subroutine: `nodecomponentblackholesimpledeserializeraw`
Description: Deserialize the contents of a simple implementation of the blackHole component from raw (binary) file.

Code lines: 12
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponentblackholesimpledeserializevalues
Description: Deserialize evolvable properties of a simple implementation of the blackHole component from array.
Code lines: 16
Contained by: module galacticus_nodes

subroutine: nodecomponentblackholesimplefinalize
Description: Finalize a simple implementation of the blackHole component.
Code lines: 13
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponentblackholesimpleinitialize
Description: Initialize a simple member of the blackHole component.
Code lines: 9
Contained by: module galacticus_nodes

function: nodecomponentblackholesimpleisactive
Description: Return true if the simple implementation of the blackHole component is the active choice.
Code lines: 8
Contained by: module galacticus_nodes

function: nodecomponentblackholesimplenamefromindex
Description: Return the name of the property of given index for a simple implementation of the blackHole component class.
Code lines: 24
Contained by: module galacticus_nodes
Modules used: iso_varying_string

subroutine: nodecomponentblackholesimpleserializeascii
Description: Serialize the contents of a simple implementation of the blackHole component to ASCII.
Code lines: 19
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string string_handling

function: nodecomponentblackholesimpleserializecount
Description: Return a count of the serialization of a simple implementation of the blackHole component.
Code lines: 22
Contained by: module galacticus_nodes

subroutine: nodecomponentblackholesimpleserializeoffsets
Description: Compute offsets into serialization arrays for a simple implementation of the blackHole component.
Code lines: 28
Contained by: module galacticus_nodes

subroutine: `nodecomponentblackholesimpleserializeraw`
Description: Serialize the contents of a simple implementation of the blackHole component to raw (binary) file.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentblackholesimpleserializevalues`
Description: Serialize evolvable properties of a `simple` implementation of the `blackHole` component to array.
Code lines: 16
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentblackholesimpleserializexml`
Description: Serialize the contents of a simple implementation of the blackHole component to XML.
Code lines: 16
Contained by: module `galacticus_nodes`

function: `nodecomponentblackholesimplesizeof`
Description: Return the size in bytes of a simple implementation of the blackHole component.
Code lines: 9
Contained by: module `galacticus_nodes`

function: `nodecomponentblackholesimpletype`
Description: Returns the type name for the simple implementation of the blackHole component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `nodecomponentblackholesizeof`
Description: Return the size in bytes of a nodeComponentBlackHole component.
Code lines: 9
Contained by: module `galacticus_nodes`

type: `nodecomponentblackholestandard`
Description: Class for the standard implementation of the blackHole component.
Code lines: 310
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentblackholestandardbuilder`
Description: Build a `standard` implementation of the `blackHole` component from a supplied XML definition.
Code lines: 42
Contained by: module `galacticus_nodes`
Modules used: `fox_dom` `galacticus_error` `memory_management`

subroutine: `nodecomponentblackholestandarddeserializeraw`
Description: Deserialize the contents of a standard implementation of the blackHole component from raw (binary) file.
Code lines: 15

Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponentblackholestandarddeserializevalues
Description: Deserialize evolvable properties of a standard implementation of the blackHole component from array.
Code lines: 20
Contained by: module galacticus_nodes

subroutine: nodecomponentblackholestandardfinalize
Description: Finalize a standard implementation of the blackHole component.
Code lines: 13
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponentblackholestandardinitialize
Description: Initialize a standard member of the blackHole component.
Code lines: 12
Contained by: module galacticus_nodes

function: nodecomponentblackholestandardisactive
Description: Return true if the standard implementation of the blackHole component is the active choice.
Code lines: 8
Contained by: module galacticus_nodes

function: nodecomponentblackholestandardnamefromindex
Description: Return the name of the property of given index for a standard implementation of the blackHole component class.
Code lines: 31
Contained by: module galacticus_nodes
Modules used: iso_varying_string

subroutine: nodecomponentblackholestandardserializeascii
Description: Serialize the contents of a standard implementation of the blackHole component to ASCII.
Code lines: 28
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string string_handling

function: nodecomponentblackholestandardserializecount
Description: Return a count of the serialization of a standard implementation of the blackHole component.
Code lines: 30
Contained by: module galacticus_nodes

subroutine: nodecomponentblackholestandardserializeoffsets
Description: Compute offsets into serialization arrays for a standard implementation of the blackHole component.
Code lines: 42
Contained by: module galacticus_nodes

subroutine: `nodecomponentblackholestandardserializeraw`
Description: Serialize the contents of a standard implementation of the blackHole component to raw (binary) file.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: `nodecomponentblackholestandardserializevalues`
Description: Serialize evolvable properties of a `standard` implementation of the `blackHole` component to array.
Code lines: 20
Contained by: module galacticus_nodes

subroutine: `nodecomponentblackholestandardserializexml`
Description: Serialize the contents of a standard implementation of the blackHole component to XML.
Code lines: 22
Contained by: module galacticus_nodes

function: `nodecomponentblackholestandardsizeof`
Description: Return the size in bytes of a standard implementation of the blackHole component.
Code lines: 9
Contained by: module galacticus_nodes

function: `nodecomponentblackholestandardtype`
Description: Returns the type name for the standard implementation of the blackHole component class.
Code lines: 13
Contained by: module galacticus_nodes

function: `nodecomponentblackholetype`
Description: Returns the type name for the blackHole component class.
Code lines: 13
Contained by: module galacticus_nodes

type: `nodecomponentdarkmatterprofile`
Description: Type for the `darkMatterProfile` component class.
Code lines: 369
Contained by: module galacticus_nodes

subroutine: `nodecomponentdarkmatterprofilebuilder`
Description: Build a generic `darkMatterProfile` component from a supplied XML definition.
Code lines: 19
Contained by: module galacticus_nodes
Modules used: `fox_dom` galacticus_error

subroutine: `nodecomponentdarkmatterprofilecreate`
Description: Create the `darkMatterProfile` component of `self`.
Code lines: 38
Contained by: module galacticus_nodes
Modules used: galacticus_display galacticus_error

iso_varying_string string_handling

subroutine: nodecomponentdarkmatterprofiledestroy
Description: Destroy the darkMatterProfile component of self
Code lines: 15
Contained by: module galacticus_nodes

subroutine: nodecomponentdarkmatterprofiledumpascii
Description: Dump the content of a darkMatterProfile component.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string

subroutine: nodecomponentdarkmatterprofilefinalize
Description: Finalize a generic darkMatterProfile component.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: nodecomponentdarkmatterprofileinitialize
Description: Initialize a generic darkMatterProfile component.
Code lines: 15
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: nodecomponentdarkmatterprofilemove
Description: Move instances of the darkMatterProfile component, from one node to another.
Code lines: 98
Contained by: module galacticus_nodes
Modules used: galacticus_error

type: nodecomponentdarkmatterprofilenull
Description: Class for the null implementation of the darkMatterProfile component.
Code lines: 124
Contained by: module galacticus_nodes

subroutine: nodecomponentdarkmatterprofilenullbuilder
Description: Build a null implementation of the darkMatterProfile component from a supplied XML definition.
Code lines: 18
Contained by: module galacticus_nodes
Modules used: fox_dom galacticus_error memory_management

subroutine: nodecomponentdarkmatterprofilenulldeserializeraw
Description: Deserialize the contents of a null implementation of the darkMatterProfile component from raw (binary) file.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: `nodecomponentdarkmatterprofilenulldeserializevalues`
Description: Deserialize evolvable properties of a null implementation of the darkMatterProfile component from array.
Code lines: 17
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentdarkmatterprofilenullfinalize`
Description: Finalize a `null` implementation of the `darkMatterProfile` component.
Code lines: 13
Contained by: module `galacticus_nodes`
Modules used: `memory_management`

subroutine: `nodecomponentdarkmatterprofilenullinitialize`
Description: Initialize a `null` member of the `darkMatterProfile` component.
Code lines: 11
Contained by: module `galacticus_nodes`

function: `nodecomponentdarkmatterprofilenullisactive`
Description: Return true if the null implementation of the darkMatterProfile component is the active choice.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `nodecomponentdarkmatterprofilenullnamefromindex`
Description: Return the name of the property of given index for a `null` implementation of the `darkMatterProfile` component class.
Code lines: 21
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

subroutine: `nodecomponentdarkmatterprofilenullserializeascii`
Description: Serialize the contents of a null implementation of the darkMatterProfile component to ASCII.
Code lines: 16
Contained by: module `galacticus_nodes`
Modules used: `galacticus_display` `iso_varying_string` `string_handling`

function: `nodecomponentdarkmatterprofilenullserializecount`
Description: Return a count of the serialization of a null implementation of the darkMatterProfile component.
Code lines: 15
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentdarkmatterprofilenullserializeoffsets`
Description: Compute offsets into serialization arrays for a `null` implementation of the `darkMatterProfile` component.
Code lines: 22
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentdarkmatterprofilenullserializeraw`
Description: Serialize the contents of a null implementation of the darkMatterProfile component to raw (binary) file.
Code lines: 14
Contained by: module galacticus_nodes

subroutine: `nodecomponentdarkmatterprofilenullserializevalues`
Description: Serialize evolvable properties of a `null` implementation of the `darkMatterProfile` component to array.
Code lines: 17
Contained by: module galacticus_nodes

subroutine: `nodecomponentdarkmatterprofilenullserializexml`
Description: Serialize the contents of a null implementation of the darkMatterProfile component to XML.
Code lines: 14
Contained by: module galacticus_nodes

function: `nodecomponentdarkmatterprofilenullsizeof`
Description: Return the size in bytes of a null implementation of the darkMatterProfile component.
Code lines: 9
Contained by: module galacticus_nodes

function: `nodecomponentdarkmatterprofilenulltype`
Description: Returns the type name for the null implementation of the darkMatterProfile component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: `nodecomponentdarkmatterprofileremove`
Description: Remove an instance of the `darkMatterProfile` component from a node.
Code lines: 58
Contained by: module galacticus_nodes
Modules used: galacticus_error

type: `nodecomponentdarkmatterprofilescale`
Description: Class for the scale implementation of the darkMatterProfile component.
Code lines: 232
Contained by: module galacticus_nodes

subroutine: `nodecomponentdarkmatterprofilescalebuilder`
Description: Build a `scale` implementation of the `darkMatterProfile` component from a supplied XML definition.
Code lines: 36
Contained by: module galacticus_nodes
Modules used: `fox_dom` galacticus_error memory_management

subroutine: `nodecomponentdarkmatterprofilescaledeserializeraw`

Description: Deserialize the contents of a scale implementation of the darkMatterProfile component from raw (binary) file.
Code lines: 14
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponentdarkmatterprofilescaledeserializevalues
Description: Deserialize evolvable properties of a scale implementation of the darkMatterProfile component from array.
Code lines: 16
Contained by: module galacticus_nodes

subroutine: nodecomponentdarkmatterprofilescalefinalize
Description: Finalize a scale implementation of the darkMatterProfile component.
Code lines: 13
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponentdarkmatterprofilescaleinitialize
Description: Initialize a scale member of the darkMatterProfile component.
Code lines: 11
Contained by: module galacticus_nodes

function: nodecomponentdarkmatterprofilescaleisactive
Description: Return true if the scale implementation of the darkMatterProfile component is the active choice.
Code lines: 8
Contained by: module galacticus_nodes

function: nodecomponentdarkmatterprofilescalenamefromindex
Description: Return the name of the property of given index for a scale implementation of the darkMatterProfile component class.
Code lines: 24
Contained by: module galacticus_nodes
Modules used: iso_varying_string

type: nodecomponentdarkmatterprofilescalepreset
Description: Class for the scalePreset implementation of the darkMatterProfile component.
Code lines: 194
Contained by: module galacticus_nodes

subroutine: nodecomponentdarkmatterprofilescalepresetbuilder
Description: Build a scalePreset implementation of the darkMatterProfile component from a supplied XML definition.
Code lines: 30
Contained by: module galacticus_nodes
Modules used: fox_dom galacticus_error memory_management

subroutine: nodecomponentdarkmatterprofilescalepresetdeserializeraw

Description: Deserialize the contents of a scalePreset implementation of the darkMatterProfile component from raw (binary) file.
Code lines: 13
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponentdarkmatterprofilescalepresetdeserializevalues
Description: Deserialize evolvable properties of a scalePreset implementation of the darkMatterProfile component from array.
Code lines: 16
Contained by: module galacticus_nodes

subroutine: nodecomponentdarkmatterprofilescalepresetfinalize
Description: Finalize a scalePreset implementation of the darkMatterProfile component.
Code lines: 13
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponentdarkmatterprofilescalepresetinitialize
Description: Initialize a scalePreset member of the darkMatterProfile component.
Code lines: 10
Contained by: module galacticus_nodes

function: nodecomponentdarkmatterprofilescalepresetisactive
Description: Return true if the scalePreset implementation of the darkMatterProfile component is the active choice.
Code lines: 8
Contained by: module galacticus_nodes

function: nodecomponentdarkmatterprofilescalepresetnamefromindex
Description: Return the name of the property of given index for a scalePreset implementation of the darkMatterProfile component class.
Code lines: 24
Contained by: module galacticus_nodes
Modules used: iso_varying_string

subroutine: nodecomponentdarkmatterprofilescalepresetserializeascii
Description: Serialize the contents of a scalePreset implementation of the darkMatterProfile component to ASCII.
Code lines: 22
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string string_handling

function: nodecomponentdarkmatterprofilescalepresetserializecount
Description: Return a count of the serialization of a scalePreset implementation of the darkMatterProfile component.
Code lines: 22
Contained by: module galacticus_nodes

subroutine: `nodecomponentdarkmatterprofilescalepresetserializeoffsets`
Description: Compute offsets into serialization arrays for a `scalePreset` implementation of the `darkMatterProfile` component.
Code lines: 28
Contained by: module galacticus_nodes

subroutine: `nodecomponentdarkmatterprofilescalepresetserializeraw`
Description: Serialize the contents of a scalePreset implementation of the darkMatterProfile component to raw (binary) file.
Code lines: 11
Contained by: module galacticus_nodes

subroutine: `nodecomponentdarkmatterprofilescalepresetserializevalues`
Description: Serialize evolvable properties of a `scalePreset` implementation of the `darkMatterProfile` component to array.
Code lines: 16
Contained by: module galacticus_nodes

subroutine: `nodecomponentdarkmatterprofilescalepresetserializexml`
Description: Serialize the contents of a scalePreset implementation of the darkMatterProfile component to XML.
Code lines: 16
Contained by: module galacticus_nodes

function: `nodecomponentdarkmatterprofilescalepresetsizeof`
Description: Return the size in bytes of a scalePreset implementation of the darkMatterProfile component.
Code lines: 9
Contained by: module galacticus_nodes

function: `nodecomponentdarkmatterprofilescalepresettype`
Description: Returns the type name for the scalePreset implementation of the darkMatterProfile component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: `nodecomponentdarkmatterprofilescaleserializeascii`
Description: Serialize the contents of a scale implementation of the darkMatterProfile component to ASCII.
Code lines: 25
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string string_handling

function: `nodecomponentdarkmatterprofilescaleserializecount`
Description: Return a count of the serialization of a scale implementation of the darkMatterProfile component.
Code lines: 22
Contained by: module galacticus_nodes

subroutine: `nodecomponentdarkmatterprofilescaleserializeoffsets`

Description: Compute offsets into serialization arrays for a `scale` implementation of the `darkMatterProfile` component.
Code lines: 28
Contained by: module galacticus_nodes

subroutine: `nodecomponentdarkmatterprofilescaleserializeraw`
Description: Serialize the contents of a scale implementation of the darkMatterProfile component to raw (binary) file.
Code lines: 12
Contained by: module galacticus_nodes

subroutine: `nodecomponentdarkmatterprofilescaleserializevalues`
Description: Serialize evolvable properties of a `scale` implementation of the `darkMatterProfile` component to array.
Code lines: 16
Contained by: module galacticus_nodes

subroutine: `nodecomponentdarkmatterprofilescaleserializexml`
Description: Serialize the contents of a scale implementation of the darkMatterProfile component to XML.
Code lines: 17
Contained by: module galacticus_nodes

type: `nodecomponentdarkmatterprofilescaleshape`
Description: Class for the scaleShape implementation of the darkMatterProfile component.
Code lines: 222
Contained by: module galacticus_nodes

subroutine: `nodecomponentdarkmatterprofilescaleshapebuilder`
Description: Build a `scaleShape` implementation of the `darkMatterProfile` component from a supplied XML definition.
Code lines: 31
Contained by: module galacticus_nodes
Modules used: `fox_dom` galacticus_error memory_management

subroutine: `nodecomponentdarkmatterprofilescaleshapedeserializeraw`
Description: Deserialize the contents of a scaleShape implementation of the darkMatterProfile component from raw (binary) file.
Code lines: 14
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: `nodecomponentdarkmatterprofilescaleshapedeserializevalues`
Description: Deserialize evolvable properties of a scaleShape implementation of the darkMatterProfile component from array.
Code lines: 21
Contained by: module galacticus_nodes

subroutine: `nodecomponentdarkmatterprofilescaleshapefinalize`

Description: Finalize a `scaleShape` implementation of the `darkMatterProfile` component.
Code lines: 13
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: `nodecomponentdarkmatterprofilescaleshapeinitialize`
Description: Initialize a `scaleShape` member of the `darkMatterProfile` component.
Code lines: 11
Contained by: module galacticus_nodes

function: `nodecomponentdarkmatterprofilescaleshapeisactive`
Description: Return true if the scaleShape implementation of the darkMatterProfile component is the active choice.
Code lines: 8
Contained by: module galacticus_nodes

function: `nodecomponentdarkmatterprofilescaleshapenamefromindex`
Description: Return the name of the property of given index for a `scaleShape` implementation of the `darkMatterProfile` component class.
Code lines: 23
Contained by: module galacticus_nodes
Modules used: iso_varying_string

subroutine: `nodecomponentdarkmatterprofilescaleshapeserializeascii`
Description: Serialize the contents of a scaleShape implementation of the darkMatterProfile component to ASCII.
Code lines: 23
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string string_handling

function: `nodecomponentdarkmatterprofilescaleshapeserializecount`
Description: Return a count of the serialization of a scaleShape implementation of the darkMatterProfile component.
Code lines: 19
Contained by: module galacticus_nodes

subroutine: `nodecomponentdarkmatterprofilescaleshapeserializeoffsets`
Description: Compute offsets into serialization arrays for a `scaleShape` implementation of the `darkMatterProfile` component.
Code lines: 26
Contained by: module galacticus_nodes

subroutine: `nodecomponentdarkmatterprofilescaleshapeserializeraw`
Description: Serialize the contents of a scaleShape implementation of the darkMatterProfile component to raw (binary) file.
Code lines: 12
Contained by: module galacticus_nodes

subroutine: `nodecomponentdarkmatterprofilescaleshapeserializevalues`

Description: Serialize evolvable properties of a `scaleShape` implementation of the `darkMatterProfile` component to array.
Code lines: 21
Contained by: module galacticus_nodes

subroutine: `nodecomponentdarkmatterprofilescaleshapeserializexml`
Description: Serialize the contents of a scaleShape implementation of the darkMatterProfile component to XML.
Code lines: 17
Contained by: module galacticus_nodes

function: `nodecomponentdarkmatterprofilescaleshapesizeof`
Description: Return the size in bytes of a scaleShape implementation of the darkMatterProfile component.
Code lines: 9
Contained by: module galacticus_nodes

function: `nodecomponentdarkmatterprofilescaleshapetype`
Description: Returns the type name for the scaleShape implementation of the darkMatterProfile component class.
Code lines: 13
Contained by: module galacticus_nodes

function: `nodecomponentdarkmatterprofilescalesizeof`
Description: Return the size in bytes of a scale implementation of the darkMatterProfile component.
Code lines: 9
Contained by: module galacticus_nodes

function: `nodecomponentdarkmatterprofilescaletype`
Description: Returns the type name for the scale implementation of the darkMatterProfile component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: `nodecomponentdarkmatterprofileserializeascii`
Description: Serialize the content of a `darkMatterProfile` component to ASCII.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string

function: `nodecomponentdarkmatterprofilesizeof`
Description: Return the size in bytes of a nodeComponentDarkMatterProfile component.
Code lines: 9
Contained by: module galacticus_nodes

function: `nodecomponentdarkmatterprofiletype`
Description: Returns the type name for the darkMatterProfile component class.
Code lines: 13
Contained by: module galacticus_nodes

type: `nodecomponentdisk`
Description: Type for the `disk` component class.
Code lines: 943
Contained by: module galacticus_nodes

subroutine: `nodecomponentdiskbuilder`
Description: Build a generic `disk` component from a supplied XML definition.
Code lines: 19
Contained by: module galacticus_nodes
Modules used: fox_dom galacticus_error

subroutine: `nodecomponentdiskcreate`
Description: Create the `disk` component of `self`.
Code lines: 38
Contained by: module galacticus_nodes
Modules used: galacticus_display galacticus_error iso_varying_string string_handling

subroutine: `nodecomponentdiskdestroy`
Description: Destroy the `disk` component of `self`
Code lines: 15
Contained by: module galacticus_nodes

subroutine: `nodecomponentdiskdumpascii`
Description: Dump the content of a `disk` component.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string

subroutine: `nodecomponentdiskfinalize`
Description: Finalize a generic `disk` component.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: `nodecomponentdiskinitialize`
Description: Initialize a generic `disk` component.
Code lines: 15
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: `nodecomponentdiskmove`
Description: Move instances of the `disk` component, from one node to another.
Code lines: 98
Contained by: module galacticus_nodes
Modules used: galacticus_error

type: `nodecomponentdisknull`
Description: Class for the null implementation of the disk component.

Code lines: 124
Contained by: module galacticus_nodes

subroutine: nodecomponentdisknullbuilder
Description: Build a null implementation of the disk component from a supplied XML definition.
Code lines: 18
Contained by: module galacticus_nodes
Modules used: fox_dom galacticus_error memory_management

subroutine: nodecomponentdisknulldeserializeraw
Description: Deserialize the contents of a null implementation of the disk component from raw (binary) file.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponentdisknulldeserializevalues
Description: Deserialize evolvable properties of a null implementation of the disk component from array.
Code lines: 17
Contained by: module galacticus_nodes

subroutine: nodecomponentdisknullfinalize
Description: Finalize a null implementation of the disk component.
Code lines: 13
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponentdisknullinitialize
Description: Initialize a null member of the disk component.
Code lines: 11
Contained by: module galacticus_nodes

function: nodecomponentdisknullisactive
Description: Return true if the null implementation of the disk component is the active choice.
Code lines: 8
Contained by: module galacticus_nodes

function: nodecomponentdisknullnamefromindex
Description: Return the name of the property of given index for a null implementation of the disk component class.
Code lines: 21
Contained by: module galacticus_nodes
Modules used: iso_varying_string

subroutine: nodecomponentdisknullserializeascii
Description: Serialize the contents of a null implementation of the disk component to ASCII.
Code lines: 16
Contained by: module galacticus_nodes

Modules used: galacticus_display iso_varying_string string_handling

function: nodecomponentdisknullserializecount
Description: Return a count of the serialization of a null implementation of the disk component.
Code lines: 15
Contained by: module galacticus_nodes

subroutine: nodecomponentdisknullserializeoffsets
Description: Compute offsets into serialization arrays for a null implementation of the disk component.
Code lines: 22
Contained by: module galacticus_nodes

subroutine: nodecomponentdisknullserializeraw
Description: Serialize the contents of a null implementation of the disk component to raw (binary) file.
Code lines: 14
Contained by: module galacticus_nodes

subroutine: nodecomponentdisknullserializevalues
Description: Serialize evolvable properties of a null implementation of the disk component to array.
Code lines: 17
Contained by: module galacticus_nodes

subroutine: nodecomponentdisknullserializexml
Description: Serialize the contents of a null implementation of the disk component to XML.
Code lines: 14
Contained by: module galacticus_nodes

function: nodecomponentdisknullsizeof
Description: Return the size in bytes of a null implementation of the disk component.
Code lines: 9
Contained by: module galacticus_nodes

function: nodecomponentdisknulltype
Description: Returns the type name for the null implementation of the disk component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: nodecomponentdiskremove
Description: Remove an instance of the disk component from a node.
Code lines: 58
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: nodecomponentdiskserializeascii
Description: Serialize the content of a disk component to ASCII.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string

function: `nodecomponentdisksizeof`
Description: Return the size in bytes of a nodeComponentDisk component.
Code lines: 9
Contained by: module galacticus_nodes

type: `nodecomponentdiskstandard`
Description: Class for the standard implementation of the disk component.
Code lines: 742
Contained by: module galacticus_nodes

subroutine: `nodecomponentdiskstandardbuilder`
Description: Build a `standard` implementation of the `disk` component from a supplied XML definition.
Code lines: 96
Contained by: module galacticus_nodes
Modules used: `fox_dom` galacticus_error memory_management

subroutine: `nodecomponentdiskstandarddeserializeraw`
Description: Deserialize the contents of a standard implementation of the disk component from raw (binary) file.
Code lines: 24
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: `nodecomponentdiskstandarddeserializevalues`
Description: Deserialize evolvable properties of a standard implementation of the disk component from array.
Code lines: 57
Contained by: module galacticus_nodes

subroutine: `nodecomponentdiskstandardfinalize`
Description: Finalize a `standard` implementation of the `disk` component.
Code lines: 15
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: `nodecomponentdiskstandardinitialize`
Description: Initialize a `standard` member of the `disk` component.
Code lines: 21
Contained by: module galacticus_nodes

function: `nodecomponentdiskstandardisactive`
Description: Return true if the standard implementation of the disk component is the active choice.
Code lines: 8
Contained by: module galacticus_nodes

function: `nodecomponentdiskstandardnamefromindex`
Description: Return the name of the property of given index for a `standard` implementation of the `disk` component class.

Code lines: 84
Contained by: module galacticus_nodes
Modules used: iso_varying_string

subroutine: nodecomponentdiskstandardserializeascii
Description: Serialize the contents of a standard implementation of the disk component to ASCII.
Code lines: 60
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string string_handling

function: nodecomponentdiskstandardserializecount
Description: Return a count of the serialization of a standard implementation of the disk component.
Code lines: 91
Contained by: module galacticus_nodes

subroutine: nodecomponentdiskstandardserializeoffsets
Description: Compute offsets into serialization arrays for a standard implementation of the disk component.
Code lines: 151
Contained by: module galacticus_nodes

subroutine: nodecomponentdiskstandardserializeraw
Description: Serialize the contents of a standard implementation of the disk component to raw (binary) file.
Code lines: 22
Contained by: module galacticus_nodes

subroutine: nodecomponentdiskstandardserializevalues
Description: Serialize evolvable properties of a standard implementation of the disk component to array.
Code lines: 57
Contained by: module galacticus_nodes

subroutine: nodecomponentdiskstandardserializexml
Description: Serialize the contents of a standard implementation of the disk component to XML.
Code lines: 34
Contained by: module galacticus_nodes

function: nodecomponentdiskstandardsizeof
Description: Return the size in bytes of a standard implementation of the disk component.
Code lines: 14
Contained by: module galacticus_nodes

function: nodecomponentdiskstandardtype
Description: Returns the type name for the standard implementation of the disk component class.
Code lines: 13
Contained by: module galacticus_nodes

function: nodecomponentdisktype

Description: Returns the type name for the disk component class.
Code lines: 13
Contained by: module galacticus_nodes

type: nodecomponentdiskverysimple
Description: Class for the verySimple implementation of the disk component.
Code lines: 492
Contained by: module galacticus_nodes

subroutine: nodecomponentdiskverysimplebuilder
Description: Build a verySimple implementation of the disk component from a supplied XML definition.
Code lines: 60
Contained by: module galacticus_nodes
Modules used: fox_dom galacticus_error memory_management

subroutine: nodecomponentdiskverysimpledeserializeraw
Description: Deserialize the contents of a verySimple implementation of the disk component from raw (binary) file.
Code lines: 18
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponentdiskverysimpledeserializevalues
Description: Deserialize evolvable properties of a verySimple implementation of the disk component from array.
Code lines: 40
Contained by: module galacticus_nodes

subroutine: nodecomponentdiskverysimplefinalize
Description: Finalize a verySimple implementation of the disk component.
Code lines: 14
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponentdiskverysimpleinitialize
Description: Initialize a verySimple member of the disk component.
Code lines: 15
Contained by: module galacticus_nodes

function: nodecomponentdiskverysimpleisactive
Description: Return true if the verySimple implementation of the disk component is the active choice.
Code lines: 8
Contained by: module galacticus_nodes

function: nodecomponentdiskverysimplenamefromindex
Description: Return the name of the property of given index for a verySimple implementation of the disk component class.
Code lines: 56
Contained by: module galacticus_nodes

Modules used: iso_varying_string

subroutine: nodecomponentdiskverysimpleserializeascii
Description: Serialize the contents of a verySimple implementation of the disk component to ASCII.
Code lines: 41
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string string_handling

function: nodecomponentdiskverysimpleserializecount
Description: Return a count of the serialization of a verySimple implementation of the disk component.
Code lines: 59
Contained by: module galacticus_nodes

subroutine: nodecomponentdiskverysimpleserializeoffsets
Description: Compute offsets into serialization arrays for a verySimple implementation of the disk component.
Code lines: 95
Contained by: module galacticus_nodes

subroutine: nodecomponentdiskverysimpleserializeraw
Description: Serialize the contents of a verySimple implementation of the disk component to raw (binary) file.
Code lines: 16
Contained by: module galacticus_nodes

subroutine: nodecomponentdiskverysimpleserializevalues
Description: Serialize evolvable properties of a verySimple implementation of the disk component to array.
Code lines: 40
Contained by: module galacticus_nodes

subroutine: nodecomponentdiskverysimpleserializexml
Description: Serialize the contents of a verySimple implementation of the disk component to XML.
Code lines: 26
Contained by: module galacticus_nodes

type: nodecomponentdiskverysimplesize
Description: Class for the verySimpleSize implementation of the disk component.
Code lines: 169
Contained by: module galacticus_nodes

subroutine: nodecomponentdiskverysimplesizebuilder
Description: Build a verySimpleSize implementation of the disk component from a supplied XML definition.
Code lines: 31
Contained by: module galacticus_nodes
Modules used: fox_dom galacticus_error memory_management

subroutine: `nodecomponentdiskverysimplesizedeserializeraw`
Description: Deserialize the contents of a verySimpleSize implementation of the disk component from raw (binary) file.
Code lines: 14
Contained by: module `galacticus_nodes`
Modules used: `memory_management`

subroutine: `nodecomponentdiskverysimplesizedeserializevalues`
Description: Deserialize evolvable properties of a verySimpleSize implementation of the disk component from array.
Code lines: 17
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentdiskverysimplesizefinalize`
Description: Finalize a `verySimpleSize` implementation of the `disk` component.
Code lines: 13
Contained by: module `galacticus_nodes`
Modules used: `memory_management`

subroutine: `nodecomponentdiskverysimplesizeinitialize`
Description: Initialize a `verySimpleSize` member of the `disk` component.
Code lines: 11
Contained by: module `galacticus_nodes`

function: `nodecomponentdiskverysimplesizeisactive`
Description: Return true if the verySimpleSize implementation of the disk component is the active choice.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `nodecomponentdiskverysimplesizenamefromindex`
Description: Return the name of the property of given index for a `verySimpleSize` implementation of the `disk` component class.
Code lines: 16
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `nodecomponentdiskverysimplesizeof`
Description: Return the size in bytes of a verySimple implementation of the disk component.
Code lines: 13
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentdiskverysimplesizeserializeascii`
Description: Serialize the contents of a verySimpleSize implementation of the disk component to ASCII.
Code lines: 23
Contained by: module `galacticus_nodes`
Modules used: `galacticus_display` `iso_varying_string` `string_handling`

function: `nodecomponentdiskverysimplesizeserializecount`
Description: Return a count of the serialization of a verySimpleSize implementation of the disk component.
Code lines: 13
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentdiskverysimplesizeserializeoffsets`
Description: Compute offsets into serialization arrays for a `verySimpleSize` implementation of the `disk` component.
Code lines: 15
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentdiskverysimplesizeserializeraw`
Description: Serialize the contents of a verySimpleSize implementation of the disk component to raw (binary) file.
Code lines: 12
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentdiskverysimplesizeserializevalues`
Description: Serialize evolvable properties of a `verySimpleSize` implementation of the `disk` component to array.
Code lines: 17
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentdiskverysimplesizeserializexml`
Description: Serialize the contents of a verySimpleSize implementation of the disk component to XML.
Code lines: 18
Contained by: module `galacticus_nodes`

function: `nodecomponentdiskverysimplesizesizeof`
Description: Return the size in bytes of a verySimpleSize implementation of the disk component.
Code lines: 9
Contained by: module `galacticus_nodes`

function: `nodecomponentdiskverysimplesizetype`
Description: Returns the type name for the verySimpleSize implementation of the disk component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `nodecomponentdiskverysimpletype`
Description: Returns the type name for the verySimple implementation of the disk component class.
Code lines: 13
Contained by: module `galacticus_nodes`

type: `nodecomponentdynamicsstatistics`
Description: Type for the `dynamicsStatistics` component class.
Code lines: 229
Contained by: module `galacticus_nodes`

type: `nodecomponentdynamicsstatisticsbars`
Description: Class for the bars implementation of the dynamicsStatistics component.
Code lines: 183
Contained by: module galacticus_nodes

subroutine: `nodecomponentdynamicsstatisticsbarsbuilder`
Description: Build a `bars` implementation of the `dynamicsStatistics` component from a supplied XML definition.
Code lines: 43
Contained by: module galacticus_nodes
Modules used: `fox_dom` galacticus_error memory_management

subroutine: `nodecomponentdynamicsstatisticsbarsdeserializeraw`
Description: Deserialize the contents of a bars implementation of the dynamicsStatistics component from raw (binary) file.
Code lines: 31
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: `nodecomponentdynamicsstatisticsbarsdeserializevalues`
Description: Deserialize evolvable properties of a bars implementation of the dynamicsStatistics component from array.
Code lines: 17
Contained by: module galacticus_nodes

subroutine: `nodecomponentdynamicsstatisticsbarsfinalize`
Description: Finalize a `bars` implementation of the `dynamicsStatistics` component.
Code lines: 13
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: `nodecomponentdynamicsstatisticsbarsinitialize`
Description: Initialize a `bars` member of the `dynamicsStatistics` component.
Code lines: 14
Contained by: module galacticus_nodes

function: `nodecomponentdynamicsstatisticsbarsisactive`
Description: Return true if the bars implementation of the dynamicsStatistics component is the active choice.
Code lines: 8
Contained by: module galacticus_nodes

function: `nodecomponentdynamicsstatisticsbarsnamefromindex`
Description: Return the name of the property of given index for a `bars` implementation of the `dynamicsStatistics` component class.
Code lines: 21
Contained by: module galacticus_nodes
Modules used: iso_varying_string

subroutine: `nodecomponentdynamicsstatisticsbarsserializeascii`
Description: Serialize the contents of a bars implementation of the dynamicsStatistics component to ASCII.
Code lines: 38
Contained by: module `galacticus_nodes`
Modules used: `galacticus_display` `iso_varying_string` `string_handling`

function: `nodecomponentdynamicsstatisticsbarsserializecount`
Description: Return a count of the serialization of a bars implementation of the dynamicsStatistics component.
Code lines: 15
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentdynamicsstatisticsbarsserializeoffsets`
Description: Compute offsets into serialization arrays for a `bars` implementation of the `dynamicsStatistics` component.
Code lines: 22
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentdynamicsstatisticsbarsserializeraw`
Description: Serialize the contents of a bars implementation of the dynamicsStatistics component to raw (binary) file.
Code lines: 24
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentdynamicsstatisticsbarsserializevalues`
Description: Serialize evolvable properties of a `bars` implementation of the `dynamicsStatistics` component to array.
Code lines: 17
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentdynamicsstatisticsbarsserializexml`
Description: Serialize the contents of a bars implementation of the dynamicsStatistics component to XML.
Code lines: 24
Contained by: module `galacticus_nodes`

function: `nodecomponentdynamicsstatisticsbarssizeof`
Description: Return the size in bytes of a bars implementation of the dynamicsStatistics component.
Code lines: 12
Contained by: module `galacticus_nodes`

function: `nodecomponentdynamicsstatisticsbarstype`
Description: Returns the type name for the bars implementation of the dynamicsStatistics component class.
Code lines: 13
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentdynamicsstatisticsbuilder`
Description: Build a generic `dynamicsStatistics` component from a supplied XML definition.
Code lines: 19
Contained by: module galacticus_nodes
Modules used: `fox_dom` galacticus_error

subroutine: `nodecomponentdynamicsstatisticscreate`
Description: Create the `dynamicsStatistics` component of `self`.
Code lines: 38
Contained by: module galacticus_nodes
Modules used: galacticus_display galacticus_error iso_varying_string string_handling

subroutine: `nodecomponentdynamicsstatisticsdestroy`
Description: Destroy the `dynamicsStatistics` component of `self`
Code lines: 15
Contained by: module galacticus_nodes

subroutine: `nodecomponentdynamicsstatisticsdumpascii`
Description: Dump the content of a `dynamicsStatistics` component.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string

subroutine: `nodecomponentdynamicsstatisticsfinalize`
Description: Finalize a generic `dynamicsStatistics` component.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: `nodecomponentdynamicsstatisticsinitialize`
Description: Initialize a generic `dynamicsStatistics` component.
Code lines: 15
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: `nodecomponentdynamicsstatisticsmove`
Description: Move instances of the `dynamicsStatistics` component, from one node to another.
Code lines: 70
Contained by: module galacticus_nodes
Modules used: galacticus_error

type: `nodecomponentdynamicsstatisticsnull`
Description: Class for the null implementation of the dynamicsStatistics component.
Code lines: 124
Contained by: module galacticus_nodes

subroutine: `nodecomponentdynamicsstatisticsnullbuilder`
Description: Build a `null` implementation of the `dynamicsStatistics` component from a supplied XML definition.

Code lines: 18
Contained by: module galacticus_nodes
Modules used: fox_dom galacticus_error memory_management

subroutine: nodecomponentdynamicsstatisticsnulldeserializeraw
Description: Deserialize the contents of a null implementation of the dynamicsStatistics component from raw (binary) file.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponentdynamicsstatisticsnulldeserializevalues
Description: Deserialize evolvable properties of a null implementation of the dynamicsStatistics component from array.
Code lines: 17
Contained by: module galacticus_nodes

subroutine: nodecomponentdynamicsstatisticsnullfinalize
Description: Finalize a null implementation of the dynamicsStatistics component.
Code lines: 13
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponentdynamicsstatisticsnullinitialize
Description: Initialize a null member of the dynamicsStatistics component.
Code lines: 11
Contained by: module galacticus_nodes

function: nodecomponentdynamicsstatisticsnullisactive
Description: Return true if the null implementation of the dynamicsStatistics component is the active choice.
Code lines: 8
Contained by: module galacticus_nodes

function: nodecomponentdynamicsstatisticsnullnamefromindex
Description: Return the name of the property of given index for a null implementation of the dynamicsStatistics component class.
Code lines: 21
Contained by: module galacticus_nodes
Modules used: iso_varying_string

subroutine: nodecomponentdynamicsstatisticsnullserializeascii
Description: Serialize the contents of a null implementation of the dynamicsStatistics component to ASCII.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string string_handling

function: `nodecomponentdynamicsstatisticsnullserializecount`
Description: Return a count of the serialization of a null implementation of the dynamicsStatistics component.
Code lines: 15
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentdynamicsstatisticsnullserializeoffsets`
Description: Compute offsets into serialization arrays for a `null` implementation of the `dynamicsStatistics` component.
Code lines: 22
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentdynamicsstatisticsnullserializeraw`
Description: Serialize the contents of a null implementation of the dynamicsStatistics component to raw (binary) file.
Code lines: 14
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentdynamicsstatisticsnullserializevalues`
Description: Serialize evolvable properties of a `null` implementation of the `dynamicsStatistics` component to array.
Code lines: 17
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentdynamicsstatisticsnullserializexml`
Description: Serialize the contents of a null implementation of the dynamicsStatistics component to XML.
Code lines: 14
Contained by: module `galacticus_nodes`

function: `nodecomponentdynamicsstatisticsnullsizeof`
Description: Return the size in bytes of a null implementation of the dynamicsStatistics component.
Code lines: 9
Contained by: module `galacticus_nodes`

function: `nodecomponentdynamicsstatisticsnulltype`
Description: Returns the type name for the null implementation of the dynamicsStatistics component class.
Code lines: 13
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentdynamicsstatisticsremove`
Description: Remove an instance of the `dynamicsStatistics` component from a node.
Code lines: 42
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `nodecomponentdynamicsstatisticsserializeascii`
Description: Serialize the content of a `dynamicsStatistics` component to ASCII.

Code lines: 16
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string

function: nodecomponentdynamicsstatisticssizeof
Description: Return the size in bytes of a nodeComponentDynamicsStatistics component.
Code lines: 9
Contained by: module galacticus_nodes

function: nodecomponentdynamicsstatisticstype
Description: Returns the type name for the dynamicsStatistics component class.
Code lines: 13
Contained by: module galacticus_nodes

type: nodecomponentformationtime
Description: Type for the formationTime component class.
Code lines: 152
Contained by: module galacticus_nodes

subroutine: nodecomponentformationtimebuilder
Description: Build a generic formationTime component from a supplied XML definition.
Code lines: 19
Contained by: module galacticus_nodes
Modules used: fox_dom galacticus_error

type: nodecomponentformationtimecole2000
Description: Class for the Cole2000 implementation of the formationTime component.
Code lines: 129
Contained by: module galacticus_nodes

subroutine: nodecomponentformationtimecole2000builder
Description: Build a Cole2000 implementation of the formationTime component from a supplied XML definition.
Code lines: 24
Contained by: module galacticus_nodes
Modules used: fox_dom galacticus_error memory_management

subroutine: nodecomponentformationtimecole2000deserializeraw
Description: Deserialize the contents of a Cole2000 implementation of the formationTime component from raw (binary) file.
Code lines: 12
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponentformationtimecole2000deserializevalues
Description: Deserialize evolvable properties of a Cole2000 implementation of the formationTime component from array.
Code lines: 17
Contained by: module galacticus_nodes

subroutine: `nodecomponentformationtimecole2000finalize`
Description: Finalize a `Cole2000` implementation of the `formationTime` component.
Code lines: 13
Contained by: module `galacticus_nodes`
Modules used: `memory_management`

subroutine: `nodecomponentformationtimecole2000initialize`
Description: Initialize a `Cole2000` member of the `formationTime` component.
Code lines: 9
Contained by: module `galacticus_nodes`

function: `nodecomponentformationtimecole2000isactive`
Description: Return true if the Cole2000 implementation of the formationTime component is the active choice.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `nodecomponentformationtimecole2000namefromindex`
Description: Return the name of the property of given index for a `Cole2000` implementation of the `formationTime` component class.
Code lines: 21
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

subroutine: `nodecomponentformationtimecole2000serializeascii`
Description: Serialize the contents of a Cole2000 implementation of the formationTime component to ASCII.
Code lines: 19
Contained by: module `galacticus_nodes`
Modules used: `galacticus_display` `iso_varying_string` `string_handling`

function: `nodecomponentformationtimecole2000serializecount`
Description: Return a count of the serialization of a Cole2000 implementation of the formationTime component.
Code lines: 15
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentformationtimecole2000serializeoffsets`
Description: Compute offsets into serialization arrays for a `Cole2000` implementation of the `formationTime` component.
Code lines: 22
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentformationtimecole2000serializeraw`
Description: Serialize the contents of a Cole2000 implementation of the formationTime component to raw (binary) file.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentformationtimecole2000serializevalues`
Description: Serialize evolvable properties of a `Cole2000` implementation of the `formationTime` component to array.
Code lines: 17
Contained by: module galacticus_nodes

subroutine: `nodecomponentformationtimecole2000serializexml`
Description: Serialize the contents of a Cole2000 implementation of the formationTime component to XML.
Code lines: 15
Contained by: module galacticus_nodes

function: `nodecomponentformationtimecole2000sizeof`
Description: Return the size in bytes of a Cole2000 implementation of the formationTime component.
Code lines: 9
Contained by: module galacticus_nodes

function: `nodecomponentformationtimecole2000type`
Description: Returns the type name for the Cole2000 implementation of the formationTime component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: `nodecomponentformationtimecreate`
Description: Create the `formationTime` component of `self`.
Code lines: 38
Contained by: module galacticus_nodes
Modules used: galacticus_display, galacticus_error, iso_varying_string, string_handling

subroutine: `nodecomponentformationtimedestroy`
Description: Destroy the `formationTime` component of `self`
Code lines: 15
Contained by: module galacticus_nodes

subroutine: `nodecomponentformationtimedumpascii`
Description: Dump the content of a `formationTime` component.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: galacticus_display, iso_varying_string

subroutine: `nodecomponentformationtimefinalize`
Description: Finalize a generic `formationTime` component.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: `nodecomponentformationtimeinitialize`
Description: Initialize a generic `formationTime` component.

Code lines: 15
Contained by: module galacticus_nodes
Modules used: galacticus_error

type: nodecomponentformationtimemassfraction
Description: Class for the massFraction implementation of the formationTime component.
Code lines: 142
Contained by: module galacticus_nodes

subroutine: nodecomponentformationtimemassfractionbuilder
Description: Build a massFraction implementation of the formationTime component from a supplied XML definition.
Code lines: 24
Contained by: module galacticus_nodes
Modules used: fox_dom galacticus_error memory_management

subroutine: nodecomponentformationtimemassfractiondeserializeraw
Description: Deserialize the contents of a massFraction implementation of the formationTime component from raw (binary) file.
Code lines: 12
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponentformationtimemassfractiondeserializevalues
Description: Deserialize evolvable properties of a massFraction implementation of the formationTime component from array.
Code lines: 17
Contained by: module galacticus_nodes

subroutine: nodecomponentformationtimemassfractionfinalize
Description: Finalize a massFraction implementation of the formationTime component.
Code lines: 13
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponentformationtimemassfractioninitialize
Description: Initialize a massFraction member of the formationTime component.
Code lines: 9
Contained by: module galacticus_nodes

function: nodecomponentformationtimemassfractionisactive
Description: Return true if the massFraction implementation of the formationTime component is the active choice.
Code lines: 8
Contained by: module galacticus_nodes

function: nodecomponentformationtimemassfractionnamefromindex
Description: Return the name of the property of given index for a massFraction implementation of the formationTime component class.
Code lines: 21

Contained by: module galacticus_nodes
Modules used: iso_varying_string

subroutine: nodecomponentformationtimemassfractionserializeascii
Description: Serialize the contents of a massFraction implementation of the formationTime component to ASCII.
Code lines: 19
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string string_handling

function: nodecomponentformationtimemassfractionserializecount
Description: Return a count of the serialization of a massFraction implementation of the formationTime component.
Code lines: 15
Contained by: module galacticus_nodes

subroutine: nodecomponentformationtimemassfractionserializeoffsets
Description: Compute offsets into serialization arrays for a massFraction implementation of the formationTime component.
Code lines: 22
Contained by: module galacticus_nodes

subroutine: nodecomponentformationtimemassfractionserializeraw
Description: Serialize the contents of a massFraction implementation of the formationTime component to raw (binary) file.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: nodecomponentformationtimemassfractionserializevalues
Description: Serialize evolvable properties of a massFraction implementation of the formationTime component to array.
Code lines: 17
Contained by: module galacticus_nodes

subroutine: nodecomponentformationtimemassfractionserializexml
Description: Serialize the contents of a massFraction implementation of the formationTime component to XML.
Code lines: 15
Contained by: module galacticus_nodes

function: nodecomponentformationtimemassfractionsizeof
Description: Return the size in bytes of a massFraction implementation of the formationTime component.
Code lines: 9
Contained by: module galacticus_nodes

function: nodecomponentformationtimemassfractiontype
Description: Returns the type name for the massFraction implementation of the formationTime component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: `nodecomponentformationtimemove`
Description: Move instances of the `formationTime` component, from one node to another.
Code lines: 84
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

type: `nodecomponentformationtimenull`
Description: Class for the null implementation of the formationTime component.
Code lines: 124
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentformationtimenullbuilder`
Description: Build a `null` implementation of the `formationTime` component from a supplied XML definition.
Code lines: 18
Contained by: module `galacticus_nodes`
Modules used: `fox_dom` `galacticus_error` `memory_management`

subroutine: `nodecomponentformationtimenulldeserializeraw`
Description: Deserialize the contents of a null implementation of the formationTime component from raw (binary) file.
Code lines: 16
Contained by: module `galacticus_nodes`
Modules used: `memory_management`

subroutine: `nodecomponentformationtimenulldeserializevalues`
Description: Deserialize evolvable properties of a null implementation of the formationTime component from array.
Code lines: 17
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentformationtimenullfinalize`
Description: Finalize a `null` implementation of the `formationTime` component.
Code lines: 13
Contained by: module `galacticus_nodes`
Modules used: `memory_management`

subroutine: `nodecomponentformationtimenullinitialize`
Description: Initialize a `null` member of the `formationTime` component.
Code lines: 11
Contained by: module `galacticus_nodes`

function: `nodecomponentformationtimenullisactive`
Description: Return true if the null implementation of the formationTime component is the active choice.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `nodecomponentformationtimenullnamefromindex`
Description: Return the name of the property of given index for a `null` implementation of the `formationTime` component class.
Code lines: 21
Contained by: module galacticus_nodes
Modules used: iso_varying_string

subroutine: `nodecomponentformationtimenullserializeascii`
Description: Serialize the contents of a null implementation of the formationTime component to ASCII.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string string_handling

function: `nodecomponentformationtimenullserializecount`
Description: Return a count of the serialization of a null implementation of the formationTime component.
Code lines: 15
Contained by: module galacticus_nodes

subroutine: `nodecomponentformationtimenullserializeoffsets`
Description: Compute offsets into serialization arrays for a `null` implementation of the `formationTime` component.
Code lines: 22
Contained by: module galacticus_nodes

subroutine: `nodecomponentformationtimenullserializeraw`
Description: Serialize the contents of a null implementation of the formationTime component to raw (binary) file.
Code lines: 14
Contained by: module galacticus_nodes

subroutine: `nodecomponentformationtimenullserializevalues`
Description: Serialize evolvable properties of a `null` implementation of the `formationTime` component to array.
Code lines: 17
Contained by: module galacticus_nodes

subroutine: `nodecomponentformationtimenullserializexml`
Description: Serialize the contents of a null implementation of the formationTime component to XML.
Code lines: 14
Contained by: module galacticus_nodes

function: `nodecomponentformationtimenullsizeof`
Description: Return the size in bytes of a null implementation of the formationTime component.
Code lines: 9
Contained by: module galacticus_nodes

function: `nodecomponentformationtimenulltype`

Description: Returns the type name for the null implementation of the formationTime component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: nodecomponentformationtimeremove
Description: Remove an instance of the formationTime component from a node.
Code lines: 50
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: nodecomponentformationtimeserializeascii
Description: Serialize the content of a formationTime component to ASCII.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string

function: nodecomponentformationtimesizeof
Description: Return the size in bytes of a nodeComponentFormationTime component.
Code lines: 9
Contained by: module galacticus_nodes

function: nodecomponentformationtimetype
Description: Returns the type name for the formationTime component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: nodecomponentgeterror
Description: Function used to report errors when attempting to get a component from a node. Error reporting is handled here to avoid having the relatively expensive creation/destruction of a varying string object in the actual get functions (which are called a very large number of times).
Code lines: 16
Contained by: module galacticus_nodes
Modules used: galacticus_error iso_varying_string string_handling

type: nodecomponenthosthistory
Description: Type for the hostHistory component class.
Code lines: 145
Contained by: module galacticus_nodes

subroutine: nodecomponenthosthistorybuilder
Description: Build a generic hostHistory component from a supplied XML definition.
Code lines: 19
Contained by: module galacticus_nodes
Modules used: fox_dom galacticus_error

subroutine: nodecomponenthosthistorycreate
Description: Create the hostHistory component of self.

Code lines: 38
Contained by: module galacticus_nodes
Modules used: galacticus_display galacticus_error iso_varying_string string_handling

subroutine: nodecomponenthosthistorydestroy
Description: Destroy the hostHistory component of self
Code lines: 15
Contained by: module galacticus_nodes

subroutine: nodecomponenthosthistorydumpascii
Description: Dump the content of a hostHistory component.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string

subroutine: nodecomponenthosthistoryfinalize
Description: Finalize a generic hostHistory component.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: nodecomponenthosthistoryinitialize
Description: Initialize a generic hostHistory component.
Code lines: 15
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: nodecomponenthosthistorymove
Description: Move instances of the hostHistory component, from one node to another.
Code lines: 70
Contained by: module galacticus_nodes
Modules used: galacticus_error

type: nodecomponenthosthistorynull
Description: Class for the null implementation of the hostHistory component.
Code lines: 124
Contained by: module galacticus_nodes

subroutine: nodecomponenthosthistorynullbuilder
Description: Build a null implementation of the hostHistory component from a supplied XML definition.
Code lines: 18
Contained by: module galacticus_nodes
Modules used: fox_dom galacticus_error memory_management

subroutine: nodecomponenthosthistorynulldeserializeraw
Description: Deserialize the contents of a null implementation of the hostHistory component from raw (binary) file.
Code lines: 16

Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponenthosthistorynulldeserializevalues
Description: Deserialize evolvable properties of a null implementation of the hostHistory component from array.
Code lines: 17
Contained by: module galacticus_nodes

subroutine: nodecomponenthosthistorynullfinalize
Description: Finalize a null implementation of the hostHistory component.
Code lines: 13
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponenthosthistorynullinitialize
Description: Initialize a null member of the hostHistory component.
Code lines: 11
Contained by: module galacticus_nodes

function: nodecomponenthosthistorynullisactive
Description: Return true if the null implementation of the hostHistory component is the active choice.
Code lines: 8
Contained by: module galacticus_nodes

function: nodecomponenthosthistorynullnamefromindex
Description: Return the name of the property of given index for a null implementation of the hostHistory component class.
Code lines: 21
Contained by: module galacticus_nodes
Modules used: iso_varying_string

subroutine: nodecomponenthosthistorynullserializeascii
Description: Serialize the contents of a null implementation of the hostHistory component to ASCII.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string string_handling

function: nodecomponenthosthistorynullserializecount
Description: Return a count of the serialization of a null implementation of the hostHistory component.
Code lines: 15
Contained by: module galacticus_nodes

subroutine: nodecomponenthosthistorynullserializeoffsets
Description: Compute offsets into serialization arrays for a null implementation of the hostHistory component.
Code lines: 22
Contained by: module galacticus_nodes

subroutine: `nodecomponenthosthistorynullserializeraw`
Description: Serialize the contents of a null implementation of the hostHistory component to raw (binary) file.
Code lines: 14
Contained by: module galacticus_nodes

subroutine: `nodecomponenthosthistorynullserializevalues`
Description: Serialize evolvable properties of a `null` implementation of the `hostHistory` component to array.
Code lines: 17
Contained by: module galacticus_nodes

subroutine: `nodecomponenthosthistorynullserializexml`
Description: Serialize the contents of a null implementation of the hostHistory component to XML.
Code lines: 14
Contained by: module galacticus_nodes

function: `nodecomponenthosthistorynullsizeof`
Description: Return the size in bytes of a null implementation of the hostHistory component.
Code lines: 9
Contained by: module galacticus_nodes

function: `nodecomponenthosthistorynulltype`
Description: Returns the type name for the null implementation of the hostHistory component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: `nodecomponenthosthistoryremove`
Description: Remove an instance of the `hostHistory` component from a node.
Code lines: 42
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: `nodecomponenthosthistoryserializeascii`
Description: Serialize the content of a `hostHistory` component to ASCII.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string

function: `nodecomponenthosthistorysizeof`
Description: Return the size in bytes of a nodeComponentHostHistory component.
Code lines: 9
Contained by: module galacticus_nodes

type: `nodecomponenthosthistorystandard`
Description: Class for the standard implementation of the hostHistory component.
Code lines: 142
Contained by: module galacticus_nodes

subroutine: `nodecomponenthosthistorystandardbuilder`
Description: Build a `standard` implementation of the `hostHistory` component from a supplied XML definition.
Code lines: 24
Contained by: module galacticus_nodes
Modules used: `fox_dom` galacticus_error memory_management

subroutine: `nodecomponenthosthistorystandarddeserializeraw`
Description: Deserialize the contents of a standard implementation of the hostHistory component from raw (binary) file.
Code lines: 12
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: `nodecomponenthosthistorystandarddeserializevalues`
Description: Deserialize evolvable properties of a standard implementation of the hostHistory component from array.
Code lines: 17
Contained by: module galacticus_nodes

subroutine: `nodecomponenthosthistorystandardfinalize`
Description: Finalize a `standard` implementation of the `hostHistory` component.
Code lines: 13
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: `nodecomponenthosthistorystandardinitialize`
Description: Initialize a `standard` member of the `hostHistory` component.
Code lines: 9
Contained by: module galacticus_nodes

function: `nodecomponenthosthistorystandardisactive`
Description: Return true if the standard implementation of the hostHistory component is the active choice.
Code lines: 8
Contained by: module galacticus_nodes

function: `nodecomponenthosthistorystandardnamefromindex`
Description: Return the name of the property of given index for a `standard` implementation of the `hostHistory` component class.
Code lines: 21
Contained by: module galacticus_nodes
Modules used: iso_varying_string

subroutine: `nodecomponenthosthistorystandardserializeascii`
Description: Serialize the contents of a standard implementation of the hostHistory component to ASCII.
Code lines: 19

Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string string_handling

function: nodecomponenthosthistorystandardserializecount
Description: Return a count of the serialization of a standard implementation of the hostHistory component.
Code lines: 15
Contained by: module galacticus_nodes

subroutine: nodecomponenthosthistorystandardserializeoffsets
Description: Compute offsets into serialization arrays for a standard implementation of the hostHistory component.
Code lines: 22
Contained by: module galacticus_nodes

subroutine: nodecomponenthosthistorystandardserializeraw
Description: Serialize the contents of a standard implementation of the hostHistory component to raw (binary) file.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: nodecomponenthosthistorystandardserializevalues
Description: Serialize evolvable properties of a standard implementation of the hostHistory component to array.
Code lines: 17
Contained by: module galacticus_nodes

subroutine: nodecomponenthosthistorystandardserializexml
Description: Serialize the contents of a standard implementation of the hostHistory component to XML.
Code lines: 15
Contained by: module galacticus_nodes

function: nodecomponenthosthistorystandardsizeof
Description: Return the size in bytes of a standard implementation of the hostHistory component.
Code lines: 9
Contained by: module galacticus_nodes

function: nodecomponenthosthistorystandardtype
Description: Returns the type name for the standard implementation of the hostHistory component class.
Code lines: 13
Contained by: module galacticus_nodes

function: nodecomponenthosthistorytype
Description: Returns the type name for the hostHistory component class.
Code lines: 13
Contained by: module galacticus_nodes

type: nodecomponenthothalo

Description: Type for the `hotHalo` component class.
Code lines: 1629
Contained by: module galacticus_nodes

subroutine: `nodecomponenthothalobuilder`
Description: Build a generic `hotHalo` component from a supplied XML definition.
Code lines: 19
Contained by: module galacticus_nodes
Modules used: `fox_dom` galacticus_error

type: `nodecomponenthothalocoldmode`
Description: Class for the coldMode implementation of the hotHalo component.
Code lines: 333
Contained by: module galacticus_nodes

subroutine: `nodecomponenthothalocoldmodebuilder`
Description: Build a `coldMode` implementation of the `hotHalo` component from a supplied XML definition.
Code lines: 37
Contained by: module galacticus_nodes
Modules used: `fox_dom` galacticus_error memory_management

subroutine: `nodecomponenthothalocoldmodedeserializeraw`
Description: Deserialize the contents of a coldMode implementation of the hotHalo component from raw (binary) file.
Code lines: 15
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: `nodecomponenthothalocoldmodedeserializevalues`
Description: Deserialize evolvable properties of a coldMode implementation of the hotHalo component from array.
Code lines: 30
Contained by: module galacticus_nodes

subroutine: `nodecomponenthothalocoldmodefinalize`
Description: Finalize a `coldMode` implementation of the `hotHalo` component.
Code lines: 11
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: `nodecomponenthothalocoldmodeinitialize`
Description: Initialize a `coldMode` member of the `hotHalo` component.
Code lines: 12
Contained by: module galacticus_nodes

function: `nodecomponenthothalocoldmodeisactive`
Description: Return true if the coldMode implementation of the hotHalo component is the active choice.

Code lines: 8
Contained by: module galacticus_nodes

function: nodecomponenthothalocoldmodenamefromindex
Description: Return the name of the property of given index for a coldMode implementation of the hotHalo component class.
Code lines: 37
Contained by: module galacticus_nodes
Modules used: iso_varying_string

subroutine: nodecomponenthothalocoldmodeserializeascii
Description: Serialize the contents of a coldMode implementation of the hotHalo component to ASCII.
Code lines: 27
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string string_handling

function: nodecomponenthothalocoldmodeserializecount
Description: Return a count of the serialization of a coldMode implementation of the hotHalo component.
Code lines: 35
Contained by: module galacticus_nodes

subroutine: nodecomponenthothalocoldmodeserializeoffsets
Description: Compute offsets into serialization arrays for a coldMode implementation of the hotHalo component.
Code lines: 54
Contained by: module galacticus_nodes

subroutine: nodecomponenthothalocoldmodeserializeraw
Description: Serialize the contents of a coldMode implementation of the hotHalo component to raw (binary) file.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: nodecomponenthothalocoldmodeserializevalues
Description: Serialize evolvable properties of a coldMode implementation of the hotHalo component to array.
Code lines: 30
Contained by: module galacticus_nodes

subroutine: nodecomponenthothalocoldmodeserializexml
Description: Serialize the contents of a coldMode implementation of the hotHalo component to XML.
Code lines: 20
Contained by: module galacticus_nodes

function: nodecomponenthothalocoldmodesizeof
Description: Return the size in bytes of a coldMode implementation of the hotHalo component.
Code lines: 10
Contained by: module galacticus_nodes

function: `nodecomponenthothalocoldmodetype`
Description: Returns the type name for the coldMode implementation of the hotHalo component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: `nodecomponenthothalocreate`
Description: Create the `hotHalo` component of `self`.
Code lines: 38
Contained by: module galacticus_nodes
Modules used: galacticus_display, galacticus_error, iso_varying_string, string_handling

subroutine: `nodecomponenthothalodestroy`
Description: Destroy the `hotHalo` component of `self`
Code lines: 15
Contained by: module galacticus_nodes

subroutine: `nodecomponenthothalodumpascii`
Description: Dump the content of a `hotHalo` component.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: galacticus_display, iso_varying_string

subroutine: `nodecomponenthothalofinalize`
Description: Finalize a generic `hotHalo` component.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: `nodecomponenthothaloinitialize`
Description: Initialize a generic `hotHalo` component.
Code lines: 15
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: `nodecomponenthothalomove`
Description: Move instances of the `hotHalo` component, from one node to another.
Code lines: 126
Contained by: module galacticus_nodes
Modules used: galacticus_error

type: `nodecomponenthothalonull`
Description: Class for the null implementation of the hotHalo component.
Code lines: 124
Contained by: module galacticus_nodes

subroutine: `nodecomponenthothalonullbuilder`
Description: Build a `null` implementation of the `hotHalo` component from a supplied XML definition.
Code lines: 18

Contained by: module galacticus_nodes
Modules used: fox_dom galacticus_error memory_management

subroutine: nodecomponenthothalonulldeserializeraw
Description: Deserialize the contents of a null implementation of the hotHalo component from raw (binary) file.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponenthothalonulldeserializevalues
Description: Deserialize evolvable properties of a null implementation of the hotHalo component from array.
Code lines: 17
Contained by: module galacticus_nodes

subroutine: nodecomponenthothalonullfinalize
Description: Finalize a null implementation of the hotHalo component.
Code lines: 13
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponenthothalonullinitialize
Description: Initialize a null member of the hotHalo component.
Code lines: 11
Contained by: module galacticus_nodes

function: nodecomponenthothalonullisactive
Description: Return true if the null implementation of the hotHalo component is the active choice.
Code lines: 8
Contained by: module galacticus_nodes

function: nodecomponenthothalonullnamefromindex
Description: Return the name of the property of given index for a null implementation of the hotHalo component class.
Code lines: 21
Contained by: module galacticus_nodes
Modules used: iso_varying_string

subroutine: nodecomponenthothalonullserializeascii
Description: Serialize the contents of a null implementation of the hotHalo component to ASCII.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string string_handling

function: nodecomponenthothalonullserializecount
Description: Return a count of the serialization of a null implementation of the hotHalo component.

Code lines: 15
Contained by: module galacticus_nodes

subroutine: nodecomponenthothalonullserializeoffsets
Description: Compute offsets into serialization arrays for a null implementation of the hotHalo component.
Code lines: 22
Contained by: module galacticus_nodes

subroutine: nodecomponenthothalonullserializeraw
Description: Serialize the contents of a null implementation of the hotHalo component to raw (binary) file.
Code lines: 14
Contained by: module galacticus_nodes

subroutine: nodecomponenthothalonullserializevalues
Description: Serialize evolvable properties of a null implementation of the hotHalo component to array.
Code lines: 17
Contained by: module galacticus_nodes

subroutine: nodecomponenthothalonullserializexml
Description: Serialize the contents of a null implementation of the hotHalo component to XML.
Code lines: 14
Contained by: module galacticus_nodes

function: nodecomponenthothalonullsizeof
Description: Return the size in bytes of a null implementation of the hotHalo component.
Code lines: 9
Contained by: module galacticus_nodes

function: nodecomponenthothalonulltype
Description: Returns the type name for the null implementation of the hotHalo component class.
Code lines: 13
Contained by: module galacticus_nodes

type: nodecomponenthothalooutflowtracking
Description: Class for the outflowTracking implementation of the hotHalo component.
Code lines: 279
Contained by: module galacticus_nodes

subroutine: nodecomponenthothalooutflowtrackingbuilder
Description: Build a outflowTracking implementation of the hotHalo component from a supplied XML definition.
Code lines: 31
Contained by: module galacticus_nodes
Modules used: fox_dom galacticus_error memory_management

subroutine: nodecomponenthothalooutflowtrackingdeserializeraw
Description: Deserialize the contents of a outflowTracking implementation of the hotHalo component from raw (binary) file.

Code lines: 14
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponenthothalooutflowtrackingdeserializevalues
Description: Deserialize evolvable properties of a outflowTracking implementation of the hotHalo component from array.
Code lines: 26
Contained by: module galacticus_nodes

subroutine: nodecomponenthothalooutflowtrackingfinalize
Description: Finalize a outflowTracking implementation of the hotHalo component.
Code lines: 11
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponenthothalooutflowtrackinginitialize
Description: Initialize a outflowTracking member of the hotHalo component.
Code lines: 11
Contained by: module galacticus_nodes

function: nodecomponenthothalooutflowtrackingisactive
Description: Return true if the outflowTracking implementation of the hotHalo component is the active choice.
Code lines: 8
Contained by: module galacticus_nodes

function: nodecomponenthothalooutflowtrackingnamefromindex
Description: Return the name of the property of given index for a outflowTracking implementation of the hotHalo component class.
Code lines: 30
Contained by: module galacticus_nodes
Modules used: iso_varying_string

subroutine: nodecomponenthothalooutflowtrackingserializeascii
Description: Serialize the contents of a outflowTracking implementation of the hotHalo component to ASCII.
Code lines: 24
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string string_handling

function: nodecomponenthothalooutflowtrackingserializecount
Description: Return a count of the serialization of a outflowTracking implementation of the hotHalo component.
Code lines: 27
Contained by: module galacticus_nodes

subroutine: nodecomponenthothalooutflowtrackingserializeoffsets
Description: Compute offsets into serialization arrays for a outflowTracking implementation of the hotHalo component.

Code lines: 40
Contained by: module galacticus_nodes

subroutine: nodecomponenthothalooutflowtrackingserializeraw
Description: Serialize the contents of a outflowTracking implementation of the hotHalo component to raw (binary) file.
Code lines: 12
Contained by: module galacticus_nodes

subroutine: nodecomponenthothalooutflowtrackingserializevalues
Description: Serialize evolvable properties of a outflowTracking implementation of the hotHalo component to array.
Code lines: 26
Contained by: module galacticus_nodes

subroutine: nodecomponenthothalooutflowtrackingserializexml
Description: Serialize the contents of a outflowTracking implementation of the hotHalo component to XML.
Code lines: 18
Contained by: module galacticus_nodes

function: nodecomponenthothalooutflowtrackingsizeof
Description: Return the size in bytes of a outflowTracking implementation of the hotHalo component.
Code lines: 10
Contained by: module galacticus_nodes

function: nodecomponenthothalooutflowtrackingtype
Description: Returns the type name for the outflowTracking implementation of the hotHalo component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: nodecomponenthothaloremove
Description: Remove an instance of the hotHalo component from a node.
Code lines: 74
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: nodecomponenthothaloserializeascii
Description: Serialize the content of a hotHalo component to ASCII.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string

function: nodecomponenthothalosizeof
Description: Return the size in bytes of a nodeComponentHotHalo component.
Code lines: 9
Contained by: module galacticus_nodes

type: `nodecomponenthothalostandard`
Description: Class for the standard implementation of the hotHalo component.
Code lines: 922
Contained by: module galacticus_nodes

subroutine: `nodecomponenthothalostandardbuilder`
Description: Build a `standard` implementation of the `hotHalo` component from a supplied XML definition.
Code lines: 90
Contained by: module galacticus_nodes
Modules used: `fox_dom` galacticus_error memory_management

subroutine: `nodecomponenthothalostandarddeserializeraw`
Description: Deserialize the contents of a standard implementation of the hotHalo component from raw (binary) file.
Code lines: 23
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: `nodecomponenthothalostandarddeserializevalues`
Description: Deserialize evolvable properties of a standard implementation of the hotHalo component from array.
Code lines: 60
Contained by: module galacticus_nodes

subroutine: `nodecomponenthothalostandardfinalize`
Description: Finalize a `standard` implementation of the `hotHalo` component.
Code lines: 14
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: `nodecomponenthothalostandardinitialize`
Description: Initialize a `standard` member of the `hotHalo` component.
Code lines: 20
Contained by: module galacticus_nodes

function: `nodecomponenthothalostandardisactive`
Description: Return true if the standard implementation of the hotHalo component is the active choice.
Code lines: 8
Contained by: module galacticus_nodes

function: `nodecomponenthothalostandardnamefromindex`
Description: Return the name of the property of given index for a `standard` implementation of the `hotHalo` component class.
Code lines: 91
Contained by: module galacticus_nodes
Modules used: iso_varying_string

subroutine: `nodecomponenthothalostandardserializeascii`
Description: Serialize the contents of a standard implementation of the hotHalo component to ASCII.
Code lines: 56
Contained by: module `galacticus_nodes`
Modules used: `galacticus_display` `iso_varying_string` `string_handling`

function: `nodecomponenthothalostandardserializecount`
Description: Return a count of the serialization of a standard implementation of the hotHalo component.
Code lines: 99
Contained by: module `galacticus_nodes`

subroutine: `nodecomponenthothalostandardserializeoffsets`
Description: Compute offsets into serialization arrays for a `standard` implementation of the `hotHalo` component.
Code lines: 165
Contained by: module `galacticus_nodes`

subroutine: `nodecomponenthothalostandardserializeraw`
Description: Serialize the contents of a standard implementation of the hotHalo component to raw (binary) file.
Code lines: 21
Contained by: module `galacticus_nodes`

subroutine: `nodecomponenthothalostandardserializevalues`
Description: Serialize evolvable properties of a `standard` implementation of the `hotHalo` component to array.
Code lines: 60
Contained by: module `galacticus_nodes`

subroutine: `nodecomponenthothalostandardserializexml`
Description: Serialize the contents of a standard implementation of the hotHalo component to XML.
Code lines: 31
Contained by: module `galacticus_nodes`

function: `nodecomponenthothalostandardsizeof`
Description: Return the size in bytes of a standard implementation of the hotHalo component.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `nodecomponenthothalostandardtype`
Description: Returns the type name for the standard implementation of the hotHalo component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `nodecomponenthothalotype`
Description: Returns the type name for the hotHalo component class.
Code lines: 13

Contained by: module galacticus_nodes

type: nodecomponenthothaloverysimple
Description: Class for the verySimple implementation of the hotHalo component.
Code lines: 361
Contained by: module galacticus_nodes

subroutine: nodecomponenthothaloverysimplebuilder
Description: Build a verySimple implementation of the hotHalo component from a supplied XML definition.
Code lines: 36
Contained by: module galacticus_nodes
Modules used: fox_dom galacticus_error memory_management

type: nodecomponenthothaloverysimpledelayed
Description: Class for the verySimpleDelayed implementation of the hotHalo component.
Code lines: 236
Contained by: module galacticus_nodes

subroutine: nodecomponenthothaloverysimpledelayedbuilder
Description: Build a verySimpleDelayed implementation of the hotHalo component from a supplied XML definition.
Code lines: 31
Contained by: module galacticus_nodes
Modules used: fox_dom galacticus_error memory_management

subroutine: nodecomponenthothaloverysimpledelayeddeserializeraw
Description: Deserialize the contents of a verySimpleDelayed implementation of the hotHalo component from raw (binary) file.
Code lines: 14
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponenthothaloverysimpledelayeddeserializevalues
Description: Deserialize evolvable properties of a verySimpleDelayed implementation of the hotHalo component from array.
Code lines: 26
Contained by: module galacticus_nodes

subroutine: nodecomponenthothaloverysimpledelayedfinalize
Description: Finalize a verySimpleDelayed implementation of the hotHalo component.
Code lines: 11
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponenthothaloverysimpledelayedinitialize
Description: Initialize a verySimpleDelayed member of the hotHalo component.
Code lines: 11

Contained by: module galacticus_nodes

function: nodecomponenthothaloverysimpledelayedisactive
Description: Return true if the verySimpleDelayed implementation of the hotHalo component is the active choice.
Code lines: 8
Contained by: module galacticus_nodes

function: nodecomponenthothaloverysimpledelayednamefromindex
Description: Return the name of the property of given index for a verySimpleDelayed implementation of the hotHalo component class.
Code lines: 30
Contained by: module galacticus_nodes
Modules used: iso_varying_string

subroutine: nodecomponenthothaloverysimpledelayedserializeascii
Description: Serialize the contents of a verySimpleDelayed implementation of the hotHalo component to ASCII.
Code lines: 24
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string string_handling

function: nodecomponenthothaloverysimpledelayedserializecount
Description: Return a count of the serialization of a verySimpleDelayed implementation of the hotHalo component.
Code lines: 27
Contained by: module galacticus_nodes

subroutine: nodecomponenthothaloverysimpledelayedserializeoffsets
Description: Compute offsets into serialization arrays for a verySimpleDelayed implementation of the hotHalo component.
Code lines: 40
Contained by: module galacticus_nodes

subroutine: nodecomponenthothaloverysimpledelayedserializeraw
Description: Serialize the contents of a verySimpleDelayed implementation of the hotHalo component to raw (binary) file.
Code lines: 12
Contained by: module galacticus_nodes

subroutine: nodecomponenthothaloverysimpledelayedserializevalues
Description: Serialize evolvable properties of a verySimpleDelayed implementation of the hotHalo component to array.
Code lines: 26
Contained by: module galacticus_nodes

subroutine: nodecomponenthothaloverysimpledelayedserializexml
Description: Serialize the contents of a verySimpleDelayed implementation of the hotHalo component to XML.
Code lines: 18

Contained by: module galacticus_nodes

function: nodecomponenthothaloverysimpledelayedsizeof
Description: Return the size in bytes of a verySimpleDelayed implementation of the hotHalo component.
Code lines: 10
Contained by: module galacticus_nodes

function: nodecomponenthothaloverysimpledelayedtype
Description: Returns the type name for the verySimpleDelayed implementation of the hotHalo component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: nodecomponenthothaloverysimpledeserializeraw
Description: Deserialize the contents of a verySimple implementation of the hotHalo component from raw (binary) file.
Code lines: 14
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponenthothaloverysimpledeserializevalues
Description: Deserialize evolvable properties of a verySimple implementation of the hotHalo component from array.
Code lines: 25
Contained by: module galacticus_nodes

subroutine: nodecomponenthothaloverysimplefinalize
Description: Finalize a verySimple implementation of the hotHalo component.
Code lines: 11
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponenthothaloverysimpleinitialize
Description: Initialize a verySimple member of the hotHalo component.
Code lines: 11
Contained by: module galacticus_nodes

function: nodecomponenthothaloverysimpleisactive
Description: Return true if the verySimple implementation of the hotHalo component is the active choice.
Code lines: 8
Contained by: module galacticus_nodes

function: nodecomponenthothaloverysimplenamefromindex
Description: Return the name of the property of given index for a verySimple implementation of the hotHalo component class.
Code lines: 35
Contained by: module galacticus_nodes
Modules used: iso_varying_string

subroutine: nodecomponenthothaloverysimpleserializeascii

Description: Serialize the contents of a verySimple implementation of the hotHalo component to ASCII.
Code lines: 26
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string string_handling

function: nodecomponenthothaloverysimpleserializecount
Description: Return a count of the serialization of a verySimple implementation of the hotHalo component.
Code lines: 35
Contained by: module galacticus_nodes

subroutine: nodecomponenthothaloverysimpleserializeoffsets
Description: Compute offsets into serialization arrays for a verySimple implementation of the hotHalo component.
Code lines: 53
Contained by: module galacticus_nodes

subroutine: nodecomponenthothaloverysimpleserializeraw
Description: Serialize the contents of a verySimple implementation of the hotHalo component to raw (binary) file.
Code lines: 12
Contained by: module galacticus_nodes

subroutine: nodecomponenthothaloverysimpleserializevalues
Description: Serialize evolvable properties of a verySimple implementation of the hotHalo component to array.
Code lines: 25
Contained by: module galacticus_nodes

subroutine: nodecomponenthothaloverysimpleserializexml
Description: Serialize the contents of a verySimple implementation of the hotHalo component to XML.
Code lines: 19
Contained by: module galacticus_nodes

function: nodecomponenthothaloverysimplesizeof
Description: Return the size in bytes of a verySimple implementation of the hotHalo component.
Code lines: 10
Contained by: module galacticus_nodes

function: nodecomponenthothaloverysimpletype
Description: Returns the type name for the verySimple implementation of the hotHalo component class.
Code lines: 13
Contained by: module galacticus_nodes

type: nodecomponentindices
Description: Type for the indices component class.
Code lines: 145
Contained by: module galacticus_nodes

subroutine: `nodecomponentindicesbuilder`
Description: Build a generic `indices` component from a supplied XML definition.
Code lines: 19
Contained by: module galacticus_nodes
Modules used: fox_dom galacticus_error

subroutine: `nodecomponentindicescreate`
Description: Create the `indices` component of `self`.
Code lines: 38
Contained by: module galacticus_nodes
Modules used: galacticus_display galacticus_error iso_varying_string string_handling

subroutine: `nodecomponentindicesdestroy`
Description: Destroy the `indices` component of `self`
Code lines: 15
Contained by: module galacticus_nodes

subroutine: `nodecomponentindicesdumpascii`
Description: Dump the content of a `indices` component.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string

subroutine: `nodecomponentindicesfinalize`
Description: Finalize a generic `indices` component.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: `nodecomponentindicesinitialize`
Description: Initialize a generic `indices` component.
Code lines: 15
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: `nodecomponentindicesmove`
Description: Move instances of the `indices` component, from one node to another.
Code lines: 70
Contained by: module galacticus_nodes
Modules used: galacticus_error

type: `nodecomponentindicesnull`
Description: Class for the null implementation of the indices component.
Code lines: 124
Contained by: module galacticus_nodes

subroutine: `nodecomponentindicesnullbuilder`
Description: Build a `null` implementation of the `indices` component from a supplied XML definition.

Code lines: 18
Contained by: module galacticus_nodes
Modules used: fox_dom galacticus_error memory_management

subroutine: nodecomponentindicesnulldeserializeraw
Description: Deserialize the contents of a null implementation of the indices component from raw (binary) file.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponentindicesnulldeserializevalues
Description: Deserialize evolvable properties of a null implementation of the indices component from array.
Code lines: 17
Contained by: module galacticus_nodes

subroutine: nodecomponentindicesnullfinalize
Description: Finalize a null implementation of the indices component.
Code lines: 13
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponentindicesnullinitialize
Description: Initialize a null member of the indices component.
Code lines: 11
Contained by: module galacticus_nodes

function: nodecomponentindicesnullisactive
Description: Return true if the null implementation of the indices component is the active choice.
Code lines: 8
Contained by: module galacticus_nodes

function: nodecomponentindicesnullnamefromindex
Description: Return the name of the property of given index for a null implementation of the indices component class.
Code lines: 21
Contained by: module galacticus_nodes
Modules used: iso_varying_string

subroutine: nodecomponentindicesnullserializeascii
Description: Serialize the contents of a null implementation of the indices component to ASCII.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string string_handling

function: nodecomponentindicesnullserializecount

Description: Return a count of the serialization of a null implementation of the indices component.
Code lines: 15
Contained by: module galacticus_nodes

subroutine: nodecomponentindicesnullserializeoffsets
Description: Compute offsets into serialization arrays for a null implementation of the indices component.
Code lines: 22
Contained by: module galacticus_nodes

subroutine: nodecomponentindicesnullserializeraw
Description: Serialize the contents of a null implementation of the indices component to raw (binary) file.
Code lines: 14
Contained by: module galacticus_nodes

subroutine: nodecomponentindicesnullserializevalues
Description: Serialize evolvable properties of a null implementation of the indices component to array.
Code lines: 17
Contained by: module galacticus_nodes

subroutine: nodecomponentindicesnullserializexml
Description: Serialize the contents of a null implementation of the indices component to XML.
Code lines: 14
Contained by: module galacticus_nodes

function: nodecomponentindicesnullsizeof
Description: Return the size in bytes of a null implementation of the indices component.
Code lines: 9
Contained by: module galacticus_nodes

function: nodecomponentindicesnulltype
Description: Returns the type name for the null implementation of the indices component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: nodecomponentindicesremove
Description: Remove an instance of the indices component from a node.
Code lines: 42
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: nodecomponentindicesserializeascii
Description: Serialize the content of a indices component to ASCII.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string

function: nodecomponentindicessizeof
Description: Return the size in bytes of a nodeComponentIndices component.

Code lines: 9
Contained by: module galacticus_nodes

type: nodecomponentindicesstandard
Description: Class for the standard implementation of the indices component.
Code lines: 142
Contained by: module galacticus_nodes

subroutine: nodecomponentindicesstandardbuilder
Description: Build a standard implementation of the indices component from a supplied XML definition.
Code lines: 24
Contained by: module galacticus_nodes
Modules used: fox_dom galacticus_error memory_management

subroutine: nodecomponentindicesstandarddeserializeraw
Description: Deserialize the contents of a standard implementation of the indices component from raw (binary) file.
Code lines: 12
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponentindicesstandarddeserializevalues
Description: Deserialize evolvable properties of a standard implementation of the indices component from array.
Code lines: 17
Contained by: module galacticus_nodes

subroutine: nodecomponentindicesstandardfinalize
Description: Finalize a standard implementation of the indices component.
Code lines: 13
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponentindicesstandardinitialize
Description: Initialize a standard member of the indices component.
Code lines: 9
Contained by: module galacticus_nodes

function: nodecomponentindicesstandardisactive
Description: Return true if the standard implementation of the indices component is the active choice.
Code lines: 8
Contained by: module galacticus_nodes

function: nodecomponentindicesstandardnamefromindex
Description: Return the name of the property of given index for a standard implementation of the indices component class.
Code lines: 21
Contained by: module galacticus_nodes

Modules used: iso_varying_string

subroutine: nodecomponentindicesstandardserializeascii
Description: Serialize the contents of a standard implementation of the indices component to ASCII.
Code lines: 19
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string string_handling

function: nodecomponentindicesstandardserializecount
Description: Return a count of the serialization of a standard implementation of the indices component.
Code lines: 15
Contained by: module galacticus_nodes

subroutine: nodecomponentindicesstandardserializeoffsets
Description: Compute offsets into serialization arrays for a standard implementation of the indices component.
Code lines: 22
Contained by: module galacticus_nodes

subroutine: nodecomponentindicesstandardserializeraw
Description: Serialize the contents of a standard implementation of the indices component to raw (binary) file.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: nodecomponentindicesstandardserializevalues
Description: Serialize evolvable properties of a standard implementation of the indices component to array.
Code lines: 17
Contained by: module galacticus_nodes

subroutine: nodecomponentindicesstandardserializexml
Description: Serialize the contents of a standard implementation of the indices component to XML.
Code lines: 15
Contained by: module galacticus_nodes

function: nodecomponentindicesstandardsizeof
Description: Return the size in bytes of a standard implementation of the indices component.
Code lines: 9
Contained by: module galacticus_nodes

function: nodecomponentindicesstandardtype
Description: Returns the type name for the standard implementation of the indices component class.
Code lines: 13
Contained by: module galacticus_nodes

function: nodecomponentindicestype
Description: Returns the type name for the indices component class.

Code lines: 13
Contained by: module galacticus_nodes

type: nodecomponentinteroutput
Description: Type for the interOutput component class.
Code lines: 229
Contained by: module galacticus_nodes

subroutine: nodecomponentinteroutputbuilder
Description: Build a generic interOutput component from a supplied XML definition.
Code lines: 19
Contained by: module galacticus_nodes
Modules used: fox_dom galacticus_error

subroutine: nodecomponentinteroutputcreate
Description: Create the interOutput component of self.
Code lines: 38
Contained by: module galacticus_nodes
Modules used: galacticus_display galacticus_error iso_varying_string string_handling

subroutine: nodecomponentinteroutputdestroy
Description: Destroy the interOutput component of self
Code lines: 15
Contained by: module galacticus_nodes

subroutine: nodecomponentinteroutputdumpascii
Description: Dump the content of a interOutput component.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string

subroutine: nodecomponentinteroutputfinalize
Description: Finalize a generic interOutput component.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: nodecomponentinteroutputinitialize
Description: Initialize a generic interOutput component.
Code lines: 15
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: nodecomponentinteroutputmove
Description: Move instances of the interOutput component, from one node to another.
Code lines: 70
Contained by: module galacticus_nodes
Modules used: galacticus_error

type: `nodecomponentinteroutputnull`
Description: Class for the null implementation of the interOutput component.
Code lines: 124
Contained by: module galacticus_nodes

subroutine: `nodecomponentinteroutputnullbuilder`
Description: Build a `null` implementation of the `interOutput` component from a supplied XML definition.
Code lines: 18
Contained by: module galacticus_nodes
Modules used: `fox_dom` galacticus_error memory_management

subroutine: `nodecomponentinteroutputnulldeserializeraw`
Description: Deserialize the contents of a null implementation of the interOutput component from raw (binary) file.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: `nodecomponentinteroutputnulldeserializevalues`
Description: Deserialize evolvable properties of a null implementation of the interOutput component from array.
Code lines: 17
Contained by: module galacticus_nodes

subroutine: `nodecomponentinteroutputnullfinalize`
Description: Finalize a `null` implementation of the `interOutput` component.
Code lines: 13
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: `nodecomponentinteroutputnullinitialize`
Description: Initialize a `null` member of the `interOutput` component.
Code lines: 11
Contained by: module galacticus_nodes

function: `nodecomponentinteroutputnullisactive`
Description: Return true if the null implementation of the interOutput component is the active choice.
Code lines: 8
Contained by: module galacticus_nodes

function: `nodecomponentinteroutputnullnamefromindex`
Description: Return the name of the property of given index for a `null` implementation of the `interOutput` component class.
Code lines: 21
Contained by: module galacticus_nodes
Modules used: iso_varying_string

subroutine: `nodecomponentinteroutputnullserializeascii`
Description: Serialize the contents of a null implementation of the interOutput component to ASCII.
Code lines: 16
Contained by: module `galacticus_nodes`
Modules used: `galacticus_display` `iso_varying_string` `string_handling`

function: `nodecomponentinteroutputnullserializecount`
Description: Return a count of the serialization of a null implementation of the interOutput component.
Code lines: 15
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentinteroutputnullserializeoffsets`
Description: Compute offsets into serialization arrays for a `null` implementation of the `interOutput` component.
Code lines: 22
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentinteroutputnullserializeraw`
Description: Serialize the contents of a null implementation of the interOutput component to raw (binary) file.
Code lines: 14
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentinteroutputnullserializevalues`
Description: Serialize evolvable properties of a `null` implementation of the `interOutput` component to array.
Code lines: 17
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentinteroutputnullserializexml`
Description: Serialize the contents of a null implementation of the interOutput component to XML.
Code lines: 14
Contained by: module `galacticus_nodes`

function: `nodecomponentinteroutputnullsizeof`
Description: Return the size in bytes of a null implementation of the interOutput component.
Code lines: 9
Contained by: module `galacticus_nodes`

function: `nodecomponentinteroutputnulltype`
Description: Returns the type name for the null implementation of the interOutput component class.
Code lines: 13
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentinteroutputremove`
Description: Remove an instance of the `interOutput` component from a node.
Code lines: 42

Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: nodecomponentinteroutputserializeascii
Description: Serialize the content of a interOutput component to ASCII.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string

function: nodecomponentinteroutputsizeof
Description: Return the size in bytes of a nodeComponentInterOutput component.
Code lines: 9
Contained by: module galacticus_nodes

type: nodecomponentinteroutputstandard
Description: Class for the standard implementation of the interOutput component.
Code lines: 229
Contained by: module galacticus_nodes

subroutine: nodecomponentinteroutputstandardbuilder
Description: Build a standard implementation of the interOutput component from a supplied XML definition.
Code lines: 30
Contained by: module galacticus_nodes
Modules used: fox_dom galacticus_error memory_management

subroutine: nodecomponentinteroutputstandarddeserializeraw
Description: Deserialize the contents of a standard implementation of the interOutput component from raw (binary) file.
Code lines: 13
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponentinteroutputstandarddeserializevalues
Description: Deserialize evolvable properties of a standard implementation of the interOutput component from array.
Code lines: 20
Contained by: module galacticus_nodes

subroutine: nodecomponentinteroutputstandardfinalize
Description: Finalize a standard implementation of the interOutput component.
Code lines: 13
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponentinteroutputstandardinitialize
Description: Initialize a standard member of the interOutput component.
Code lines: 10

Contained by: module galacticus_nodes

function: nodecomponentinteroutputstandardisactive
Description: Return true if the standard implementation of the interOutput component is the active choice.
Code lines: 8
Contained by: module galacticus_nodes

function: nodecomponentinteroutputstandardnamefromindex
Description: Return the name of the property of given index for a standard implementation of the interOutput component class.
Code lines: 31
Contained by: module galacticus_nodes
Modules used: iso_varying_string

subroutine: nodecomponentinteroutputstandardserializeascii
Description: Serialize the contents of a standard implementation of the interOutput component to ASCII.
Code lines: 22
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string string_handling

function: nodecomponentinteroutputstandardserializecount
Description: Return a count of the serialization of a standard implementation of the interOutput component.
Code lines: 30
Contained by: module galacticus_nodes

subroutine: nodecomponentinteroutputstandardserializeoffsets
Description: Compute offsets into serialization arrays for a standard implementation of the interOutput component.
Code lines: 42
Contained by: module galacticus_nodes

subroutine: nodecomponentinteroutputstandardserializeraw
Description: Serialize the contents of a standard implementation of the interOutput component to raw (binary) file.
Code lines: 11
Contained by: module galacticus_nodes

subroutine: nodecomponentinteroutputstandardserializevalues
Description: Serialize evolvable properties of a standard implementation of the interOutput component to array.
Code lines: 20
Contained by: module galacticus_nodes

subroutine: nodecomponentinteroutputstandardserializexml
Description: Serialize the contents of a standard implementation of the interOutput component to XML.
Code lines: 16
Contained by: module galacticus_nodes

function: `nodecomponentinteroutputstandardsizeof`
Description: Return the size in bytes of a standard implementation of the interOutput component.
Code lines: 9
Contained by: module galacticus_nodes

function: `nodecomponentinteroutputstandardtype`
Description: Returns the type name for the standard implementation of the interOutput component class.
Code lines: 13
Contained by: module galacticus_nodes

function: `nodecomponentinteroutputtype`
Description: Returns the type name for the interOutput component class.
Code lines: 13
Contained by: module galacticus_nodes

type: `nodecomponentmassflowstatistics`
Description: Type for the `massFlowStatistics` component class.
Code lines: 166
Contained by: module galacticus_nodes

subroutine: `nodecomponentmassflowstatisticsbuilder`
Description: Build a generic `massFlowStatistics` component from a supplied XML definition.
Code lines: 19
Contained by: module galacticus_nodes
Modules used: `fox_dom` galacticus_error

subroutine: `nodecomponentmassflowstatisticscreate`
Description: Create the `massFlowStatistics` component of `self`.
Code lines: 38
Contained by: module galacticus_nodes
Modules used: galacticus_display galacticus_error iso_varying_string string_handling

subroutine: `nodecomponentmassflowstatisticsdestroy`
Description: Destroy the `massFlowStatistics` component of `self`
Code lines: 15
Contained by: module galacticus_nodes

subroutine: `nodecomponentmassflowstatisticsdumpascii`
Description: Dump the content of a `massFlowStatistics` component.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string

subroutine: `nodecomponentmassflowstatisticsfinalize`
Description: Finalize a generic `massFlowStatistics` component.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: `nodecomponentmassflowstatisticsinitialize`
Description: Initialize a generic `massFlowStatistics` component.
Code lines: 15
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `nodecomponentmassflowstatisticsmove`
Description: Move instances of the `massFlowStatistics` component, from one node to another.
Code lines: 70
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

type: `nodecomponentmassflowstatisticsnull`
Description: Class for the null implementation of the massFlowStatistics component.
Code lines: 124
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentmassflowstatisticsnullbuilder`
Description: Build a `null` implementation of the `massFlowStatistics` component from a supplied XML definition.
Code lines: 18
Contained by: module `galacticus_nodes`
Modules used: `fox_dom` `galacticus_error` `memory_management`

subroutine: `nodecomponentmassflowstatisticsnulldeserializeraw`
Description: Deserialize the contents of a null implementation of the massFlowStatistics component from raw (binary) file.
Code lines: 16
Contained by: module `galacticus_nodes`
Modules used: `memory_management`

subroutine: `nodecomponentmassflowstatisticsnulldeserializevalues`
Description: Deserialize evolvable properties of a null implementation of the massFlowStatistics component from array.
Code lines: 17
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentmassflowstatisticsnullfinalize`
Description: Finalize a `null` implementation of the `massFlowStatistics` component.
Code lines: 13
Contained by: module `galacticus_nodes`
Modules used: `memory_management`

subroutine: `nodecomponentmassflowstatisticsnullinitialize`
Description: Initialize a `null` member of the `massFlowStatistics` component.
Code lines: 11
Contained by: module `galacticus_nodes`

function: `nodecomponentmassflowstatisticsnullisactive`
Description: Return true if the null implementation of the massFlowStatistics component is the active choice.
Code lines: 8
Contained by: module galacticus_nodes

function: `nodecomponentmassflowstatisticsnullnamefromindex`
Description: Return the name of the property of given index for a `null` implementation of the `massFlowStatistics` component class.
Code lines: 21
Contained by: module galacticus_nodes
Modules used: iso_varying_string

subroutine: `nodecomponentmassflowstatisticsnullserializeascii`
Description: Serialize the contents of a null implementation of the massFlowStatistics component to ASCII.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string string_handling

function: `nodecomponentmassflowstatisticsnullserializecount`
Description: Return a count of the serialization of a null implementation of the massFlowStatistics component.
Code lines: 15
Contained by: module galacticus_nodes

subroutine: `nodecomponentmassflowstatisticsnullserializeoffsets`
Description: Compute offsets into serialization arrays for a `null` implementation of the `massFlowStatistics` component.
Code lines: 22
Contained by: module galacticus_nodes

subroutine: `nodecomponentmassflowstatisticsnullserializeraw`
Description: Serialize the contents of a null implementation of the massFlowStatistics component to raw (binary) file.
Code lines: 14
Contained by: module galacticus_nodes

subroutine: `nodecomponentmassflowstatisticsnullserializevalues`
Description: Serialize evolvable properties of a `null` implementation of the `massFlowStatistics` component to array.
Code lines: 17
Contained by: module galacticus_nodes

subroutine: `nodecomponentmassflowstatisticsnullserializexml`
Description: Serialize the contents of a null implementation of the massFlowStatistics component to XML.
Code lines: 14
Contained by: module galacticus_nodes

function: `nodecomponentmassflowstatisticsnullsizeof`
Description: Return the size in bytes of a null implementation of the massFlowStatistics component.
Code lines: 9
Contained by: module galacticus_nodes

function: `nodecomponentmassflowstatisticsnulltype`
Description: Returns the type name for the null implementation of the massFlowStatistics component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: `nodecomponentmassflowstatisticsremove`
Description: Remove an instance of the `massFlowStatistics` component from a node.
Code lines: 42
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: `nodecomponentmassflowstatisticsserializeascii`
Description: Serialize the content of a `massFlowStatistics` component to ASCII.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string

function: `nodecomponentmassflowstatisticssizeof`
Description: Return the size in bytes of a nodeComponentMassFlowStatistics component.
Code lines: 9
Contained by: module galacticus_nodes

type: `nodecomponentmassflowstatisticsstandard`
Description: Class for the standard implementation of the massFlowStatistics component.
Code lines: 177
Contained by: module galacticus_nodes

subroutine: `nodecomponentmassflowstatisticsstandardbuilder`
Description: Build a `standard` implementation of the `massFlowStatistics` component from a supplied XML definition.
Code lines: 24
Contained by: module galacticus_nodes
Modules used: fox_dom galacticus_error memory_management

subroutine: `nodecomponentmassflowstatisticsstandarddeserializeraw`
Description: Deserialize the contents of a standard implementation of the massFlowStatistics component from raw (binary) file.
Code lines: 12
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: `nodecomponentmassflowstatisticsstandarddeserializevalues`

Description: Deserialize evolvable properties of a standard implementation of the massFlowStatistics component from array.
Code lines: 16
Contained by: module galacticus_nodes

subroutine: nodecomponentmassflowstatisticsstandardfinalize
Description: Finalize a standard implementation of the massFlowStatistics component.
Code lines: 13
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponentmassflowstatisticsstandardinitialize
Description: Initialize a standard member of the massFlowStatistics component.
Code lines: 9
Contained by: module galacticus_nodes

function: nodecomponentmassflowstatisticsstandardisactive
Description: Return true if the standard implementation of the massFlowStatistics component is the active choice.
Code lines: 8
Contained by: module galacticus_nodes

function: nodecomponentmassflowstatisticsstandardnamefromindex
Description: Return the name of the property of given index for a standard implementation of the massFlowStatistics component class.
Code lines: 24
Contained by: module galacticus_nodes
Modules used: iso_varying_string

subroutine: nodecomponentmassflowstatisticsstandardserializeascii
Description: Serialize the contents of a standard implementation of the massFlowStatistics component to ASCII.
Code lines: 19
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string string_handling

function: nodecomponentmassflowstatisticsstandardserializecount
Description: Return a count of the serialization of a standard implementation of the massFlowStatistics component.
Code lines: 22
Contained by: module galacticus_nodes

subroutine: nodecomponentmassflowstatisticsstandardserializeoffsets
Description: Compute offsets into serialization arrays for a standard implementation of the massFlowStatistics component.
Code lines: 28
Contained by: module galacticus_nodes

subroutine: nodecomponentmassflowstatisticsstandardserializeraw

Description: Serialize the contents of a standard implementation of the massFlowStatistics component to raw (binary) file.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: nodecomponentmassflowstatisticsstandardserializevalues
Description: Serialize evolvable properties of a standard implementation of the massFlowStatistics component to array.
Code lines: 16
Contained by: module galacticus_nodes

subroutine: nodecomponentmassflowstatisticsstandardserializexml
Description: Serialize the contents of a standard implementation of the massFlowStatistics component to XML.
Code lines: 15
Contained by: module galacticus_nodes

function: nodecomponentmassflowstatisticsstandardsizeof
Description: Return the size in bytes of a standard implementation of the massFlowStatistics component.
Code lines: 9
Contained by: module galacticus_nodes

function: nodecomponentmassflowstatisticsstandardtype
Description: Returns the type name for the standard implementation of the massFlowStatistics component class.
Code lines: 13
Contained by: module galacticus_nodes

function: nodecomponentmassflowstatisticstype
Description: Returns the type name for the massFlowStatistics component class.
Code lines: 13
Contained by: module galacticus_nodes

type: nodecomponentmergingstatistics
Description: Type for the mergingStatistics component class.
Code lines: 453
Contained by: module galacticus_nodes

subroutine: nodecomponentmergingstatisticsbuilder
Description: Build a generic mergingStatistics component from a supplied XML definition.
Code lines: 19
Contained by: module galacticus_nodes
Modules used: fox_dom galacticus_error

subroutine: nodecomponentmergingstatisticscreate
Description: Create the mergingStatistics component of self.
Code lines: 38
Contained by: module galacticus_nodes
Modules used: galacticus_display galacticus_error

iso_varying_string string_handling

subroutine: nodecomponentmergingstatisticsdestroy
Description: Destroy the mergingStatistics component of self
Code lines: 15
Contained by: module galacticus_nodes

subroutine: nodecomponentmergingstatisticsdumpascii
Description: Dump the content of a mergingStatistics component.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string

subroutine: nodecomponentmergingstatisticsfinalize
Description: Finalize a generic mergingStatistics component.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: nodecomponentmergingstatisticsinitialize
Description: Initialize a generic mergingStatistics component.
Code lines: 15
Contained by: module galacticus_nodes
Modules used: galacticus_error

type: nodecomponentmergingstatisticsmajor
Description: Class for the major implementation of the mergingStatistics component.
Code lines: 142
Contained by: module galacticus_nodes

subroutine: nodecomponentmergingstatisticsmajorbuilder
Description: Build a major implementation of the mergingStatistics component from a supplied XML definition.
Code lines: 27
Contained by: module galacticus_nodes
Modules used: fox_dom galacticus_error memory_management

subroutine: nodecomponentmergingstatisticsmajordeserializeraw
Description: Deserialize the contents of a major implementation of the mergingStatistics component from raw (binary) file.
Code lines: 19
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponentmergingstatisticsmajordeserializevalues
Description: Deserialize evolvable properties of a major implementation of the mergingStatistics component from array.
Code lines: 17
Contained by: module galacticus_nodes

subroutine: `nodecomponentmergingstatisticsmajorfinalize`
Description: Finalize a `major` implementation of the `mergingStatistics` component.
Code lines: 11
Contained by: module `galacticus_nodes`
Modules used: `memory_management`

subroutine: `nodecomponentmergingstatisticsmajorinitialize`
Description: Initialize a `major` member of the `mergingStatistics` component.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `nodecomponentmergingstatisticsmajorisactive`
Description: Return true if the major implementation of the mergingStatistics component is the active choice.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `nodecomponentmergingstatisticsmajornamefromindex`
Description: Return the name of the property of given index for a `major` implementation of the `mergingStatistics` component class.
Code lines: 21
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

subroutine: `nodecomponentmergingstatisticsmajorserializeascii`
Description: Serialize the contents of a major implementation of the mergingStatistics component to ASCII.
Code lines: 24
Contained by: module `galacticus_nodes`
Modules used: `galacticus_display` `iso_varying_string` `string_handling`

function: `nodecomponentmergingstatisticsmajorserializecount`
Description: Return a count of the serialization of a major implementation of the mergingStatistics component.
Code lines: 15
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentmergingstatisticsmajorserializeoffsets`
Description: Compute offsets into serialization arrays for a `major` implementation of the `mergingStatistics` component.
Code lines: 22
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentmergingstatisticsmajorserializeraw`
Description: Serialize the contents of a major implementation of the mergingStatistics component to raw (binary) file.
Code lines: 14
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentmergingstatisticsmajorserializevalues`
Description: Serialize evolvable properties of a `major` implementation of the `mergingStatistics` component to array.
Code lines: 17
Contained by: module galacticus_nodes

subroutine: `nodecomponentmergingstatisticsmajorserializexml`
Description: Serialize the contents of a major implementation of the mergingStatistics component to XML.
Code lines: 18
Contained by: module galacticus_nodes

function: `nodecomponentmergingstatisticsmajorsizeof`
Description: Return the size in bytes of a major implementation of the mergingStatistics component.
Code lines: 10
Contained by: module galacticus_nodes

function: `nodecomponentmergingstatisticsmajortype`
Description: Returns the type name for the major implementation of the mergingStatistics component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: `nodecomponentmergingstatisticsmove`
Description: Move instances of the `mergingStatistics` component, from one node to another.
Code lines: 98
Contained by: module galacticus_nodes
Modules used: galacticus_error

type: `nodecomponentmergingstatisticsnull`
Description: Class for the null implementation of the mergingStatistics component.
Code lines: 124
Contained by: module galacticus_nodes

subroutine: `nodecomponentmergingstatisticsnullbuilder`
Description: Build a `null` implementation of the `mergingStatistics` component from a supplied XML definition.
Code lines: 18
Contained by: module galacticus_nodes
Modules used: fox_dom galacticus_error memory_management

subroutine: `nodecomponentmergingstatisticsnulldeserializeraw`
Description: Deserialize the contents of a null implementation of the mergingStatistics component from raw (binary) file.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: `nodecomponentmergingstatisticsnulldeserializevalues`
Description: Deserialize evolvable properties of a null implementation of the mergingStatistics component from array.
Code lines: 17
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentmergingstatisticsnullfinalize`
Description: Finalize a `null` implementation of the `mergingStatistics` component.
Code lines: 13
Contained by: module `galacticus_nodes`
Modules used: `memory_management`

subroutine: `nodecomponentmergingstatisticsnullinitialize`
Description: Initialize a `null` member of the `mergingStatistics` component.
Code lines: 11
Contained by: module `galacticus_nodes`

function: `nodecomponentmergingstatisticsnullisactive`
Description: Return true if the null implementation of the mergingStatistics component is the active choice.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `nodecomponentmergingstatisticsnullnamefromindex`
Description: Return the name of the property of given index for a `null` implementation of the `mergingStatistics` component class.
Code lines: 21
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

subroutine: `nodecomponentmergingstatisticsnullserializeascii`
Description: Serialize the contents of a null implementation of the mergingStatistics component to ASCII.
Code lines: 16
Contained by: module `galacticus_nodes`
Modules used: `galacticus_display` `iso_varying_string` `string_handling`

function: `nodecomponentmergingstatisticsnullserializecount`
Description: Return a count of the serialization of a null implementation of the mergingStatistics component.
Code lines: 15
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentmergingstatisticsnullserializeoffsets`
Description: Compute offsets into serialization arrays for a `null` implementation of the `mergingStatistics` component.
Code lines: 22
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentmergingstatisticsnullserializeraw`

Description: Serialize the contents of a null implementation of the mergingStatistics component to raw (binary) file.
Code lines: 14
Contained by: module galacticus_nodes

subroutine: `nodecomponentmergingstatisticsnullserializevalues`

Description: Serialize evolvable properties of a `null` implementation of the `mergingStatistics` component to array.
Code lines: 17
Contained by: module galacticus_nodes

subroutine: `nodecomponentmergingstatisticsnullserializexml`

Description: Serialize the contents of a null implementation of the mergingStatistics component to XML.
Code lines: 14
Contained by: module galacticus_nodes

function: `nodecomponentmergingstatisticsnullsizeof`

Description: Return the size in bytes of a null implementation of the mergingStatistics component.
Code lines: 9
Contained by: module galacticus_nodes

function: `nodecomponentmergingstatisticsnulltype`

Description: Returns the type name for the null implementation of the mergingStatistics component class.
Code lines: 13
Contained by: module galacticus_nodes

type: `nodecomponentmergingstatisticsrecent`

Description: Class for the recent implementation of the mergingStatistics component.
Code lines: 142
Contained by: module galacticus_nodes

subroutine: `nodecomponentmergingstatisticsrecentbuilder`

Description: Build a `recent` implementation of the `mergingStatistics` component from a supplied XML definition.
Code lines: 27
Contained by: module galacticus_nodes
Modules used: `fox_dom` galacticus_error memory_management

subroutine: `nodecomponentmergingstatisticsrecentdeserializeraw`

Description: Deserialize the contents of a recent implementation of the mergingStatistics component from raw (binary) file.
Code lines: 19
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: `nodecomponentmergingstatisticsrecentdeserializevalues`

Description: Deserialize evolvable properties of a recent implementation of the mergingStatistics component from array.
Code lines: 17
Contained by: module galacticus_nodes

subroutine: nodecomponentmergingstatisticsrecentfinalize
Description: Finalize a recent implementation of the mergingStatistics component.
Code lines: 11
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponentmergingstatisticsrecentinitialize
Description: Initialize a recent member of the mergingStatistics component.
Code lines: 12
Contained by: module galacticus_nodes
Modules used: node_component_merging_statistics_-recent_data

function: nodecomponentmergingstatisticsrecentisactive
Description: Return true if the recent implementation of the mergingStatistics component is the active choice.
Code lines: 8
Contained by: module galacticus_nodes

function: nodecomponentmergingstatisticsrecentnamefromindex
Description: Return the name of the property of given index for a recent implementation of the mergingStatistics component class.
Code lines: 21
Contained by: module galacticus_nodes
Modules used: iso_varying_string

subroutine: nodecomponentmergingstatisticsrecentserializeascii
Description: Serialize the contents of a recent implementation of the mergingStatistics component to ASCII.
Code lines: 24
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string string_handling

function: nodecomponentmergingstatisticsrecentserializecount
Description: Return a count of the serialization of a recent implementation of the mergingStatistics component.
Code lines: 15
Contained by: module galacticus_nodes

subroutine: nodecomponentmergingstatisticsrecentserializeoffsets
Description: Compute offsets into serialization arrays for a recent implementation of the mergingStatistics component.
Code lines: 22
Contained by: module galacticus_nodes

subroutine: `nodecomponentmergingstatisticsrecentserializeraw`
Description: Serialize the contents of a recent implementation of the mergingStatistics component to raw (binary) file.
Code lines: 14
Contained by: module galacticus_nodes

subroutine: `nodecomponentmergingstatisticsrecentserializevalues`
Description: Serialize evolvable properties of a `recent` implementation of the `mergingStatistics` component to array.
Code lines: 17
Contained by: module galacticus_nodes

subroutine: `nodecomponentmergingstatisticsrecentserializexml`
Description: Serialize the contents of a recent implementation of the mergingStatistics component to XML.
Code lines: 18
Contained by: module galacticus_nodes

function: `nodecomponentmergingstatisticsrecentsizeof`
Description: Return the size in bytes of a recent implementation of the mergingStatistics component.
Code lines: 10
Contained by: module galacticus_nodes

function: `nodecomponentmergingstatisticsrecenttype`
Description: Returns the type name for the recent implementation of the mergingStatistics component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: `nodecomponentmergingstatisticsremove`
Description: Remove an instance of the `mergingStatistics` component from a node.
Code lines: 58
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: `nodecomponentmergingstatisticsserializeascii`
Description: Serialize the content of a `mergingStatistics` component to ASCII.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string

function: `nodecomponentmergingstatisticssizeof`
Description: Return the size in bytes of a nodeComponentMergingStatistics component.
Code lines: 9
Contained by: module galacticus_nodes

type: `nodecomponentmergingstatisticsstandard`
Description: Class for the standard implementation of the mergingStatistics component.

Code lines: 269
Contained by: module galacticus_nodes

subroutine: nodecomponentmergingstatisticsstandardbuilder
Description: Build a standard implementation of the mergingStatistics component from a supplied XML definition.
Code lines: 54
Contained by: module galacticus_nodes
Modules used: fox_dom galacticus_error memory_management

subroutine: nodecomponentmergingstatisticsstandarddeserializeraw
Description: Deserialize the contents of a standard implementation of the mergingStatistics component from raw (binary) file.
Code lines: 17
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponentmergingstatisticsstandarddeserializevalues
Description: Deserialize evolvable properties of a standard implementation of the mergingStatistics component from array.
Code lines: 17
Contained by: module galacticus_nodes

subroutine: nodecomponentmergingstatisticsstandardfinalize
Description: Finalize a standard implementation of the mergingStatistics component.
Code lines: 13
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponentmergingstatisticsstandardinitialize
Description: Initialize a standard member of the mergingStatistics component.
Code lines: 14
Contained by: module galacticus_nodes

function: nodecomponentmergingstatisticsstandardisactive
Description: Return true if the standard implementation of the mergingStatistics component is the active choice.
Code lines: 8
Contained by: module galacticus_nodes

function: nodecomponentmergingstatisticsstandardnamefromindex
Description: Return the name of the property of given index for a standard implementation of the mergingStatistics component class.
Code lines: 21
Contained by: module galacticus_nodes
Modules used: iso_varying_string

subroutine: nodecomponentmergingstatisticsstandardserializeascii
Description: Serialize the contents of a standard implementation of the mergingStatistics component to ASCII.

Code lines: 34
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string string_handling

function: nodecomponentmergingstatisticsstandardserializecount
Description: Return a count of the serialization of a standard implementation of the mergingStatistics component.
Code lines: 15
Contained by: module galacticus_nodes

subroutine: nodecomponentmergingstatisticsstandardserializeoffsets
Description: Compute offsets into serialization arrays for a **standard** implementation of the **mergingStatistics** component.
Code lines: 22
Contained by: module galacticus_nodes

subroutine: nodecomponentmergingstatisticsstandardserializeraw
Description: Serialize the contents of a standard implementation of the mergingStatistics component to raw (binary) file.
Code lines: 15
Contained by: module galacticus_nodes

subroutine: nodecomponentmergingstatisticsstandardserializevalues
Description: Serialize evolvable properties of a **standard** implementation of the **mergingStatistics** component to array.
Code lines: 17
Contained by: module galacticus_nodes

subroutine: nodecomponentmergingstatisticsstandardserializexml
Description: Serialize the contents of a standard implementation of the mergingStatistics component to XML.
Code lines: 20
Contained by: module galacticus_nodes

function: nodecomponentmergingstatisticsstandardsizeof
Description: Return the size in bytes of a standard implementation of the mergingStatistics component.
Code lines: 9
Contained by: module galacticus_nodes

function: nodecomponentmergingstatisticsstandardtype
Description: Returns the type name for the standard implementation of the mergingStatistics component class.
Code lines: 13
Contained by: module galacticus_nodes

function: nodecomponentmergingstatisticstype
Description: Returns the type name for the mergingStatistics component class.
Code lines: 13
Contained by: module galacticus_nodes

function: `nodecomponentnamefromindex`
Description: Return the name of the property of given index for a `nodeComponent` object.
Code lines: 21
Contained by: module galacticus_nodes
Modules used: iso_varying_string

type: `nodecomponentnbody`
Description: Type for the `nBody` component class.
Code lines: 187
Contained by: module galacticus_nodes

subroutine: `nodecomponentnbodybuilder`
Description: Build a generic `nBody` component from a supplied XML definition.
Code lines: 19
Contained by: module galacticus_nodes
Modules used: fox_dom galacticus_error

subroutine: `nodecomponentnbodycreate`
Description: Create the `nBody` component of `self`.
Code lines: 38
Contained by: module galacticus_nodes
Modules used: galacticus_display galacticus_error
iso_varying_string string_handling

subroutine: `nodecomponentnbodydestroy`
Description: Destroy the `nBody` component of `self`
Code lines: 15
Contained by: module galacticus_nodes

subroutine: `nodecomponentnbodydumpascii`
Description: Dump the content of a `nBody` component.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string

subroutine: `nodecomponentnbodyfinalize`
Description: Finalize a generic `nBody` component.
Code lines: 13
Contained by: module galacticus_nodes

type: `nodecomponentnbodygeneric`
Description: Class for the generic implementation of the nBody component.
Code lines: 243
Contained by: module galacticus_nodes

subroutine: `nodecomponentnbodygenericbuilder`
Description: Build a `generic` implementation of the `nBody` component from a supplied XML definition.
Code lines: 31

Contained by: module galacticus_nodes
Modules used: fox_dom galacticus_error memory_management

subroutine: nodecomponentnbodygenericdeserializeraw
Description: Deserialize the contents of a generic implementation of the nBody component from raw (binary) file.
Code lines: 25
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponentnbodygenericdeserializevalues
Description: Deserialize evolvable properties of a generic implementation of the nBody component from array.
Code lines: 17
Contained by: module galacticus_nodes

subroutine: nodecomponentnbodygenericfinalize
Description: Finalize a generic implementation of the nBody component.
Code lines: 12
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponentnbodygenericinitialize
Description: Initialize a generic member of the nBody component.
Code lines: 12
Contained by: module galacticus_nodes

function: nodecomponentnbodygenericisactive
Description: Return true if the generic implementation of the nBody component is the active choice.
Code lines: 8
Contained by: module galacticus_nodes

function: nodecomponentnbodygenericnamefromindex
Description: Return the name of the property of given index for a generic implementation of the nBody component class.
Code lines: 21
Contained by: module galacticus_nodes
Modules used: iso_varying_string

subroutine: nodecomponentnbodygenericserializeascii
Description: Serialize the contents of a generic implementation of the nBody component to ASCII.
Code lines: 31
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string string_handling

function: nodecomponentnbodygenericserializecount
Description: Return a count of the serialization of a generic implementation of the nBody component.

Code lines: 15
Contained by: module galacticus_nodes

subroutine: nodecomponentnbodygenericserializeoffsets
Description: Compute offsets into serialization arrays for a generic implementation of the nBody component.
Code lines: 22
Contained by: module galacticus_nodes

subroutine: nodecomponentnbodygenericserializeraw
Description: Serialize the contents of a generic implementation of the nBody component to raw (binary) file.
Code lines: 19
Contained by: module galacticus_nodes

subroutine: nodecomponentnbodygenericserializevalues
Description: Serialize evolvable properties of a generic implementation of the nBody component to array.
Code lines: 17
Contained by: module galacticus_nodes

subroutine: nodecomponentnbodygenericserializexml
Description: Serialize the contents of a generic implementation of the nBody component to XML.
Code lines: 21
Contained by: module galacticus_nodes

function: nodecomponentnbodygenericsizeof
Description: Return the size in bytes of a generic implementation of the nBody component.
Code lines: 11
Contained by: module galacticus_nodes

function: nodecomponentnbodygenerictype
Description: Returns the type name for the generic implementation of the nBody component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: nodecomponentnbodyinitialize
Description: Initialize a generic nBody component.
Code lines: 15
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: nodecomponentnbodymove
Description: Move instances of the nBody component, from one node to another.
Code lines: 70
Contained by: module galacticus_nodes
Modules used: galacticus_error

type: nodecomponentnbodynull
Description: Class for the null implementation of the nBody component.

Code lines: 124
Contained by: module galacticus_nodes

subroutine: nodecomponentnbodynullbuilder
Description: Build a null implementation of the nBody component from a supplied XML definition.
Code lines: 18
Contained by: module galacticus_nodes
Modules used: fox_dom galacticus_error memory_management

subroutine: nodecomponentnbodynulldeserializeraw
Description: Deserialize the contents of a null implementation of the nBody component from raw (binary) file.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponentnbodynulldeserializevalues
Description: Deserialize evolvable properties of a null implementation of the nBody component from array.
Code lines: 17
Contained by: module galacticus_nodes

subroutine: nodecomponentnbodynullfinalize
Description: Finalize a null implementation of the nBody component.
Code lines: 13
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponentnbodynullinitialize
Description: Initialize a null member of the nBody component.
Code lines: 11
Contained by: module galacticus_nodes

function: nodecomponentnbodynullisactive
Description: Return true if the null implementation of the nBody component is the active choice.
Code lines: 8
Contained by: module galacticus_nodes

function: nodecomponentnbodynullnamefromindex
Description: Return the name of the property of given index for a null implementation of the nBody component class.
Code lines: 21
Contained by: module galacticus_nodes
Modules used: iso_varying_string

subroutine: nodecomponentnbodynullserializeascii
Description: Serialize the contents of a null implementation of the nBody component to ASCII.
Code lines: 16

Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string string_handling

function: nodecomponentnbodynullserializecount
Description: Return a count of the serialization of a null implementation of the nBody component.
Code lines: 15
Contained by: module galacticus_nodes

subroutine: nodecomponentnbodynullserializeoffsets
Description: Compute offsets into serialization arrays for a null implementation of the nBody component.
Code lines: 22
Contained by: module galacticus_nodes

subroutine: nodecomponentnbodynullserializeraw
Description: Serialize the contents of a null implementation of the nBody component to raw (binary) file.
Code lines: 14
Contained by: module galacticus_nodes

subroutine: nodecomponentnbodynullserializevalues
Description: Serialize evolvable properties of a null implementation of the nBody component to array.
Code lines: 17
Contained by: module galacticus_nodes

subroutine: nodecomponentnbodynullserializexml
Description: Serialize the contents of a null implementation of the nBody component to XML.
Code lines: 14
Contained by: module galacticus_nodes

function: nodecomponentnbodynullsizeof
Description: Return the size in bytes of a null implementation of the nBody component.
Code lines: 9
Contained by: module galacticus_nodes

function: nodecomponentnbodynulltype
Description: Returns the type name for the null implementation of the nBody component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: nodecomponentnbodyremove
Description: Remove an instance of the nBody component from a node.
Code lines: 42
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: nodecomponentnbodyserializeascii
Description: Serialize the content of a nBody component to ASCII.
Code lines: 16
Contained by: module galacticus_nodes

Modules used: galacticus_display iso_varying_string

function: nodecomponentnbodysizeof
Description: Return the size in bytes of a nodeComponentNBody component.
Code lines: 9
Contained by: module galacticus_nodes

function: nodecomponentnbodytype
Description: Returns the type name for the nBody component class.
Code lines: 13
Contained by: module galacticus_nodes

type: nodecomponentposition
Description: Type for the position component class.
Code lines: 369
Contained by: module galacticus_nodes

subroutine: nodecomponentpositionbuilder
Description: Build a generic position component from a supplied XML definition.
Code lines: 19
Contained by: module galacticus_nodes
Modules used: fox_dom galacticus_error

subroutine: nodecomponentpositioncreate
Description: Create the position component of self.
Code lines: 38
Contained by: module galacticus_nodes
Modules used: galacticus_display galacticus_error iso_varying_string string_handling

subroutine: nodecomponentpositiondestroy
Description: Destroy the position component of self
Code lines: 15
Contained by: module galacticus_nodes

subroutine: nodecomponentpositiondumpascii
Description: Dump the content of a position component.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string

subroutine: nodecomponentpositionfinalize
Description: Finalize a generic position component.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: nodecomponentpositioninitialize
Description: Initialize a generic position component.
Code lines: 15

Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: nodecomponentpositionmove
Description: Move instances of the position component, from one node to another.
Code lines: 98
Contained by: module galacticus_nodes
Modules used: galacticus_error

type: nodecomponentpositionnull
Description: Class for the null implementation of the position component.
Code lines: 124
Contained by: module galacticus_nodes

subroutine: nodecomponentpositionnullbuilder
Description: Build a null implementation of the position component from a supplied XML definition.
Code lines: 18
Contained by: module galacticus_nodes
Modules used: fox_dom galacticus_error memory_management

subroutine: nodecomponentpositionnulldeserializeraw
Description: Deserialize the contents of a null implementation of the position component from raw (binary) file.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponentpositionnulldeserializevalues
Description: Deserialize evolvable properties of a null implementation of the position component from array.
Code lines: 17
Contained by: module galacticus_nodes

subroutine: nodecomponentpositionnullfinalize
Description: Finalize a null implementation of the position component.
Code lines: 13
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponentpositionnullinitialize
Description: Initialize a null member of the position component.
Code lines: 11
Contained by: module galacticus_nodes

function: nodecomponentpositionnullisactive
Description: Return true if the null implementation of the position component is the active choice.
Code lines: 8
Contained by: module galacticus_nodes

function: `nodecomponentpositionnullnamefromindex`
Description: Return the name of the property of given index for a `null` implementation of the `position` component class.
Code lines: 21
Contained by: module galacticus_nodes
Modules used: iso_varying_string

subroutine: `nodecomponentpositionnullserializeascii`
Description: Serialize the contents of a null implementation of the position component to ASCII.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string string_handling

function: `nodecomponentpositionnullserializecount`
Description: Return a count of the serialization of a null implementation of the position component.
Code lines: 15
Contained by: module galacticus_nodes

subroutine: `nodecomponentpositionnullserializeoffsets`
Description: Compute offsets into serialization arrays for a `null` implementation of the `position` component.
Code lines: 22
Contained by: module galacticus_nodes

subroutine: `nodecomponentpositionnullserializeraw`
Description: Serialize the contents of a null implementation of the position component to raw (binary) file.
Code lines: 14
Contained by: module galacticus_nodes

subroutine: `nodecomponentpositionnullserializevalues`
Description: Serialize evolvable properties of a `null` implementation of the `position` component to array.
Code lines: 17
Contained by: module galacticus_nodes

subroutine: `nodecomponentpositionnullserializexml`
Description: Serialize the contents of a null implementation of the position component to XML.
Code lines: 14
Contained by: module galacticus_nodes

function: `nodecomponentpositionnullsizeof`
Description: Return the size in bytes of a null implementation of the position component.
Code lines: 9
Contained by: module galacticus_nodes

function: `nodecomponentpositionnulltype`
Description: Returns the type name for the null implementation of the position component class.

Code lines: 13
Contained by: module galacticus_nodes

type: nodecomponentpositionpreset
Description: Class for the preset implementation of the position component.
Code lines: 164
Contained by: module galacticus_nodes

subroutine: nodecomponentpositionpresetbuilder
Description: Build a preset implementation of the position component from a supplied XML definition.
Code lines: 41
Contained by: module galacticus_nodes
Modules used: fox_dom galacticus_error memory_management

subroutine: nodecomponentpositionpresetdeserializeraw
Description: Deserialize the contents of a preset implementation of the position component from raw (binary) file.
Code lines: 26
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponentpositionpresetdeserializevalues
Description: Deserialize evolvable properties of a preset implementation of the position component from array.
Code lines: 17
Contained by: module galacticus_nodes

subroutine: nodecomponentpositionpresetfinalize
Description: Finalize a preset implementation of the position component.
Code lines: 13
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponentpositionpresetinitialize
Description: Initialize a preset member of the position component.
Code lines: 13
Contained by: module galacticus_nodes

function: nodecomponentpositionpresetisactive
Description: Return true if the preset implementation of the position component is the active choice.
Code lines: 8
Contained by: module galacticus_nodes

function: nodecomponentpositionpresetnamefromindex
Description: Return the name of the property of given index for a preset implementation of the position component class.
Code lines: 21
Contained by: module galacticus_nodes
Modules used: iso_varying_string

type: `nodecomponentpositionpresetorphans`
Description: Class for the presetOrphans implementation of the position component.
Code lines: 245
Contained by: module galacticus_nodes

subroutine: `nodecomponentpositionpresetorphansbuilder`
Description: Build a `presetOrphans` implementation of the `position` component from a supplied XML definition.
Code lines: 58
Contained by: module galacticus_nodes
Modules used: `fox_dom` galacticus_error memory_management

subroutine: `nodecomponentpositionpresetorphansdeserializeraw`
Description: Deserialize the contents of a presetOrphans implementation of the position component from raw (binary) file.
Code lines: 39
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: `nodecomponentpositionpresetorphansdeserializevalues`
Description: Deserialize evolvable properties of a presetOrphans implementation of the position component from array.
Code lines: 17
Contained by: module galacticus_nodes

subroutine: `nodecomponentpositionpresetorphansfinalize`
Description: Finalize a `presetOrphans` implementation of the `position` component.
Code lines: 14
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: `nodecomponentpositionpresetorphansinitialize`
Description: Initialize a `presetOrphans` member of the `position` component.
Code lines: 18
Contained by: module galacticus_nodes

function: `nodecomponentpositionpresetorphansisactive`
Description: Return true if the presetOrphans implementation of the position component is the active choice.
Code lines: 8
Contained by: module galacticus_nodes

function: `nodecomponentpositionpresetorphansnamefromindex`
Description: Return the name of the property of given index for a `presetOrphans` implementation of the `position` component class.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: iso_varying_string

subroutine: `nodecomponentpositionpresetorphansserializeascii`
Description: Serialize the contents of a presetOrphans implementation of the position component to ASCII.
Code lines: 49
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string string_handling

function: `nodecomponentpositionpresetorphansserializecount`
Description: Return a count of the serialization of a presetOrphans implementation of the position component.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: `nodecomponentpositionpresetorphansserializeoffsets`
Description: Compute offsets into serialization arrays for a `presetOrphans` implementation of the `position` component.
Code lines: 15
Contained by: module galacticus_nodes

subroutine: `nodecomponentpositionpresetorphansserializeraw`
Description: Serialize the contents of a presetOrphans implementation of the position component to raw (binary) file.
Code lines: 31
Contained by: module galacticus_nodes

subroutine: `nodecomponentpositionpresetorphansserializevalues`
Description: Serialize evolvable properties of a `presetOrphans` implementation of the `position` component to array.
Code lines: 17
Contained by: module galacticus_nodes

subroutine: `nodecomponentpositionpresetorphansserializexml`
Description: Serialize the contents of a presetOrphans implementation of the position component to XML.
Code lines: 29
Contained by: module galacticus_nodes

function: `nodecomponentpositionpresetorphanssizeof`
Description: Return the size in bytes of a presetOrphans implementation of the position component.
Code lines: 13
Contained by: module galacticus_nodes

function: `nodecomponentpositionpresetorphanstype`
Description: Returns the type name for the presetOrphans implementation of the position component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: `nodecomponentpositionpresetserializeascii`

Description: Serialize the contents of a preset implementation of the position component to ASCII.
Code lines: 35
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string string_handling

function: nodecomponentpositionpresetserializecount
Description: Return a count of the serialization of a preset implementation of the position component.
Code lines: 15
Contained by: module galacticus_nodes

subroutine: nodecomponentpositionpresetserializeoffsets
Description: Compute offsets into serialization arrays for a preset implementation of the position component.
Code lines: 22
Contained by: module galacticus_nodes

subroutine: nodecomponentpositionpresetserializeraw
Description: Serialize the contents of a preset implementation of the position component to raw (binary) file.
Code lines: 20
Contained by: module galacticus_nodes

subroutine: nodecomponentpositionpresetserializevalues
Description: Serialize evolvable properties of a preset implementation of the position component to array.
Code lines: 17
Contained by: module galacticus_nodes

subroutine: nodecomponentpositionpresetserializexml
Description: Serialize the contents of a preset implementation of the position component to XML.
Code lines: 23
Contained by: module galacticus_nodes

function: nodecomponentpositionpresetsizeof
Description: Return the size in bytes of a preset implementation of the position component.
Code lines: 12
Contained by: module galacticus_nodes

function: nodecomponentpositionpresettype
Description: Returns the type name for the preset implementation of the position component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: nodecomponentpositionremove
Description: Remove an instance of the position component from a node.
Code lines: 58
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: `nodecomponentpositionserializeascii`
Description: Serialize the content of a `position` component to ASCII.
Code lines: 16
Contained by: module `galacticus_nodes`
Modules used: `galacticus_display` `iso_varying_string`

function: `nodecomponentpositionsizeof`
Description: Return the size in bytes of a nodeComponentPosition component.
Code lines: 9
Contained by: module `galacticus_nodes`

type: `nodecomponentpositiontracedarkmatter`
Description: Class for the traceDarkMatter implementation of the position component.
Code lines: 142
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentpositiontracedarkmatterbuilder`
Description: Build a `traceDarkMatter` implementation of the `position` component from a supplied XML definition.
Code lines: 27
Contained by: module `galacticus_nodes`
Modules used: `fox_dom` `galacticus_error` `memory_management`

subroutine: `nodecomponentpositiontracedarkmatterdeserializeraw`
Description: Deserialize the contents of a traceDarkMatter implementation of the position component from raw (binary) file.
Code lines: 19
Contained by: module `galacticus_nodes`
Modules used: `memory_management`

subroutine: `nodecomponentpositiontracedarkmatterdeserializevalues`
Description: Deserialize evolvable properties of a traceDarkMatter implementation of the position component from array.
Code lines: 17
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentpositiontracedarkmatterfinalize`
Description: Finalize a `traceDarkMatter` implementation of the `position` component.
Code lines: 11
Contained by: module `galacticus_nodes`
Modules used: `memory_management`

subroutine: `nodecomponentpositiontracedarkmatterinitialize`
Description: Initialize a `traceDarkMatter` member of the `position` component.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `nodecomponentpositiontracedarkmatterisactive`

Description: Return true if the traceDarkMatter implementation of the position component is the active choice.
Code lines: 8
Contained by: module galacticus_nodes

function: `nodecomponentpositiontracedarkmatternamefromindex`

Description: Return the name of the property of given index for a `traceDarkMatter` implementation of the `position` component class.
Code lines: 21
Contained by: module galacticus_nodes
Modules used: iso_varying_string

subroutine: `nodecomponentpositiontracedarkmatterserializeascii`

Description: Serialize the contents of a traceDarkMatter implementation of the position component to ASCII.
Code lines: 24
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string string_handling

function: `nodecomponentpositiontracedarkmatterserializecount`

Description: Return a count of the serialization of a traceDarkMatter implementation of the position component.
Code lines: 15
Contained by: module galacticus_nodes

subroutine: `nodecomponentpositiontracedarkmatterserializeoffsets`

Description: Compute offsets into serialization arrays for a `traceDarkMatter` implementation of the `position` component.
Code lines: 22
Contained by: module galacticus_nodes

subroutine: `nodecomponentpositiontracedarkmatterserializeraw`

Description: Serialize the contents of a traceDarkMatter implementation of the position component to raw (binary) file.
Code lines: 14
Contained by: module galacticus_nodes

subroutine: `nodecomponentpositiontracedarkmatterserializevalues`

Description: Serialize evolvable properties of a `traceDarkMatter` implementation of the `position` component to array.
Code lines: 17
Contained by: module galacticus_nodes

subroutine: `nodecomponentpositiontracedarkmatterserializexml`

Description: Serialize the contents of a traceDarkMatter implementation of the position component to XML.
Code lines: 18
Contained by: module galacticus_nodes

function: `nodecomponentpositiontracedarkmattersizeof`
Description: Return the size in bytes of a traceDarkMatter implementation of the position component.
Code lines: 10
Contained by: module galacticus_nodes

function: `nodecomponentpositiontracedarkmattertype`
Description: Returns the type name for the traceDarkMatter implementation of the position component class.
Code lines: 13
Contained by: module galacticus_nodes

function: `nodecomponentpositiontype`
Description: Returns the type name for the position component class.
Code lines: 13
Contained by: module galacticus_nodes

type: `nodecomponentsatellite`
Description: Type for the `satellite` component class.
Code lines: 747
Contained by: module galacticus_nodes

subroutine: `nodecomponentsatellitebuilder`
Description: Build a generic `satellite` component from a supplied XML definition.
Code lines: 19
Contained by: module galacticus_nodes
Modules used: `fox_dom` galacticus_error

subroutine: `nodecomponentsatellitecreate`
Description: Create the `satellite` component of `self`.
Code lines: 38
Contained by: module galacticus_nodes
Modules used: galacticus_display galacticus_error iso_varying_string string_handling

subroutine: `nodecomponentsatellitedestroy`
Description: Destroy the `satellite` component of `self`
Code lines: 15
Contained by: module galacticus_nodes

subroutine: `nodecomponentsatellitedumpascii`
Description: Dump the content of a `satellite` component.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string

subroutine: `nodecomponentsatellitefinalize`
Description: Finalize a generic `satellite` component.
Code lines: 13

Contained by: module galacticus_nodes

subroutine: nodecomponentsatelliteinitialize
Description: Initialize a generic satellite component.
Code lines: 15
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: nodecomponentsatellitemove
Description: Move instances of the satellite component, from one node to another.
Code lines: 112
Contained by: module galacticus_nodes
Modules used: galacticus_error

type: nodecomponentsatellitenull
Description: Class for the null implementation of the satellite component.
Code lines: 124
Contained by: module galacticus_nodes

subroutine: nodecomponentsatellitenullbuilder
Description: Build a null implementation of the satellite component from a supplied XML definition.
Code lines: 18
Contained by: module galacticus_nodes
Modules used: fox_dom galacticus_error memory_management

subroutine: nodecomponentsatellitenulldeserializeraw
Description: Deserialize the contents of a null implementation of the satellite component from raw (binary) file.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponentsatellitenulldeserializevalues
Description: Deserialize evolvable properties of a null implementation of the satellite component from array.
Code lines: 17
Contained by: module galacticus_nodes

subroutine: nodecomponentsatellitenullfinalize
Description: Finalize a null implementation of the satellite component.
Code lines: 13
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponentsatellitenullinitialize
Description: Initialize a null member of the satellite component.
Code lines: 11
Contained by: module galacticus_nodes

function: `nodecomponentsatellitenullisactive`
Description: Return true if the null implementation of the satellite component is the active choice.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `nodecomponentsatellitenullnamefromindex`
Description: Return the name of the property of given index for a `null` implementation of the `satellite` component class.
Code lines: 21
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

subroutine: `nodecomponentsatellitenullserializeascii`
Description: Serialize the contents of a null implementation of the satellite component to ASCII.
Code lines: 16
Contained by: module `galacticus_nodes`
Modules used: `galacticus_display` `iso_varying_string` `string_handling`

function: `nodecomponentsatellitenullserializecount`
Description: Return a count of the serialization of a null implementation of the satellite component.
Code lines: 15
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentsatellitenullserializeoffsets`
Description: Compute offsets into serialization arrays for a `null` implementation of the `satellite` component.
Code lines: 22
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentsatellitenullserializeraw`
Description: Serialize the contents of a null implementation of the satellite component to raw (binary) file.
Code lines: 14
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentsatellitenullserializevalues`
Description: Serialize evolvable properties of a `null` implementation of the `satellite` component to array.
Code lines: 17
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentsatellitenullserializexml`
Description: Serialize the contents of a null implementation of the satellite component to XML.
Code lines: 14
Contained by: module `galacticus_nodes`

function: `nodecomponentsatellitenullsizeof`

Description: Return the size in bytes of a null implementation of the satellite component.
Code lines: 9
Contained by: module galacticus_nodes

function: nodecomponentsatellitenulltype
Description: Returns the type name for the null implementation of the satellite component class.
Code lines: 13
Contained by: module galacticus_nodes

type: nodecomponentsatelliteorbiting
Description: Class for the orbiting implementation of the satellite component.
Code lines: 443
Contained by: module galacticus_nodes

subroutine: nodecomponentsatelliteorbitingbuilder
Description: Build a orbiting implementation of the satellite component from a supplied XML definition.
Code lines: 65
Contained by: module galacticus_nodes
Modules used: fox_dom galacticus_error memory_management

subroutine: nodecomponentsatelliteorbitingdeserializeraw
Description: Deserialize the contents of a orbiting implementation of the satellite component from raw (binary) file.
Code lines: 30
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponentsatelliteorbitingdeserializevalues
Description: Deserialize evolvable properties of a orbiting implementation of the satellite component from array.
Code lines: 39
Contained by: module galacticus_nodes

subroutine: nodecomponentsatelliteorbitingfinalize
Description: Finalize a orbiting implementation of the satellite component.
Code lines: 14
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponentsatelliteorbitinginitialize
Description: Initialize a orbiting member of the satellite component.
Code lines: 19
Contained by: module galacticus_nodes

function: nodecomponentsatelliteorbitingisactive
Description: Return true if the orbiting implementation of the satellite component is the active choice.
Code lines: 8

Contained by: module galacticus_nodes

function: nodecomponentsatelliteorbitingnamefromindex
Description: Return the name of the property of given index for a orbiting implementation of the satellite component class.
Code lines: 49
Contained by: module galacticus_nodes
Modules used: iso_varying_string

subroutine: nodecomponentsatelliteorbitingserializeascii
Description: Serialize the contents of a orbiting implementation of the satellite component to ASCII.
Code lines: 48
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string string_handling

function: nodecomponentsatelliteorbitingserializecount
Description: Return a count of the serialization of a orbiting implementation of the satellite component.
Code lines: 51
Contained by: module galacticus_nodes

subroutine: nodecomponentsatelliteorbitingserializeoffsets
Description: Compute offsets into serialization arrays for a orbiting implementation of the satellite component.
Code lines: 81
Contained by: module galacticus_nodes

subroutine: nodecomponentsatelliteorbitingserializeraw
Description: Serialize the contents of a orbiting implementation of the satellite component to raw (binary) file.
Code lines: 24
Contained by: module galacticus_nodes

subroutine: nodecomponentsatelliteorbitingserializevalues
Description: Serialize evolvable properties of a orbiting implementation of the satellite component to array.
Code lines: 39
Contained by: module galacticus_nodes

subroutine: nodecomponentsatelliteorbitingserializexml
Description: Serialize the contents of a orbiting implementation of the satellite component to XML.
Code lines: 29
Contained by: module galacticus_nodes

function: nodecomponentsatelliteorbitingsizeof
Description: Return the size in bytes of a orbiting implementation of the satellite component.
Code lines: 13
Contained by: module galacticus_nodes

function: `nodecomponentsatelliteorbitingtype`
Description: Returns the type name for the orbiting implementation of the satellite component class.
Code lines: 13
Contained by: module galacticus_nodes

type: `nodecomponentsatellitepreset`
Description: Class for the preset implementation of the satellite component.
Code lines: 221
Contained by: module galacticus_nodes

subroutine: `nodecomponentsatellitepresetbuilder`
Description: Build a `preset` implementation of the `satellite` component from a supplied XML definition.
Code lines: 54
Contained by: module galacticus_nodes
Modules used: fox_dom galacticus_error memory_management

subroutine: `nodecomponentsatellitepresetdeserializeraw`
Description: Deserialize the contents of a preset implementation of the satellite component from raw (binary) file.
Code lines: 17
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: `nodecomponentsatellitepresetdeserializevalues`
Description: Deserialize evolvable properties of a preset implementation of the satellite component from array.
Code lines: 17
Contained by: module galacticus_nodes

subroutine: `nodecomponentsatellitepresetfinalize`
Description: Finalize a `preset` implementation of the `satellite` component.
Code lines: 13
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: `nodecomponentsatellitepresetinitialize`
Description: Initialize a `preset` member of the `satellite` component.
Code lines: 14
Contained by: module galacticus_nodes

function: `nodecomponentsatellitepresetisactive`
Description: Return true if the preset implementation of the satellite component is the active choice.
Code lines: 8
Contained by: module galacticus_nodes

function: `nodecomponentsatellitepresetnamefromindex`

Description: Return the name of the property of given index for a `preset` implementation of the `satellite` component class.
Code lines: 21
Contained by: module galacticus_nodes
Modules used: iso_varying_string

subroutine: `nodecomponentsatellitepresetserializeascii`
Description: Serialize the contents of a preset implementation of the satellite component to ASCII.
Code lines: 37
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string string_handling

function: `nodecomponentsatellitepresetserializecount`
Description: Return a count of the serialization of a preset implementation of the satellite component.
Code lines: 15
Contained by: module galacticus_nodes

subroutine: `nodecomponentsatellitepresetserializeoffsets`
Description: Compute offsets into serialization arrays for a `preset` implementation of the `satellite` component.
Code lines: 22
Contained by: module galacticus_nodes

subroutine: `nodecomponentsatellitepresetserializeraw`
Description: Serialize the contents of a preset implementation of the satellite component to raw (binary) file.
Code lines: 15
Contained by: module galacticus_nodes

subroutine: `nodecomponentsatellitepresetserializevalues`
Description: Serialize evolvable properties of a `preset` implementation of the `satellite` component to array.
Code lines: 17
Contained by: module galacticus_nodes

subroutine: `nodecomponentsatellitepresetserializexml`
Description: Serialize the contents of a preset implementation of the satellite component to XML.
Code lines: 25
Contained by: module galacticus_nodes

function: `nodecomponentsatellitepresetsizeof`
Description: Return the size in bytes of a preset implementation of the satellite component.
Code lines: 12
Contained by: module galacticus_nodes

function: `nodecomponentsatellitepresettype`
Description: Returns the type name for the preset implementation of the satellite component class.
Code lines: 13

Contained by: module galacticus_nodes

subroutine: nodecomponentsatelliteremove
Description: Remove an instance of the satellite component from a node.
Code lines: 66
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: nodecomponentsatelliteserializeascii
Description: Serialize the content of a satellite component to ASCII.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string

function: nodecomponentsatellitesizeof
Description: Return the size in bytes of a nodeComponentSatellite component.
Code lines: 9
Contained by: module galacticus_nodes

type: nodecomponentsatellitestandard
Description: Class for the standard implementation of the satellite component.
Code lines: 293
Contained by: module galacticus_nodes

subroutine: nodecomponentsatellitestandardbuilder
Description: Build a standard implementation of the satellite component from a supplied XML definition.
Code lines: 36
Contained by: module galacticus_nodes
Modules used: fox_dom galacticus_error memory_management

subroutine: nodecomponentsatellitestandarddeserializeraw
Description: Deserialize the contents of a standard implementation of the satellite component from raw (binary) file.
Code lines: 14
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponentsatellitestandarddeserializevalues
Description: Deserialize evolvable properties of a standard implementation of the satellite component from array.
Code lines: 20
Contained by: module galacticus_nodes

subroutine: nodecomponentsatellitestandardfinalize
Description: Finalize a standard implementation of the satellite component.
Code lines: 11
Contained by: module galacticus_nodes

Modules used: memory_management

subroutine: nodecomponentsatellitestandardinitialize
Description: Initialize a standard member of the satellite component.
Code lines: 13
Contained by: module galacticus_nodes

function: nodecomponentsatellitestandardisactive
Description: Return true if the standard implementation of the satellite component is the active choice.
Code lines: 8
Contained by: module galacticus_nodes

function: nodecomponentsatellitestandardnamefromindex
Description: Return the name of the property of given index for a standard implementation of the satellite component class.
Code lines: 31
Contained by: module galacticus_nodes
Modules used: iso_varying_string

subroutine: nodecomponentsatellitestandardserializeascii
Description: Serialize the contents of a standard implementation of the satellite component to ASCII.
Code lines: 26
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string string_handling

function: nodecomponentsatellitestandardserializecount
Description: Return a count of the serialization of a standard implementation of the satellite component.
Code lines: 30
Contained by: module galacticus_nodes

subroutine: nodecomponentsatellitestandardserializeoffsets
Description: Compute offsets into serialization arrays for a standard implementation of the satellite component.
Code lines: 42
Contained by: module galacticus_nodes

subroutine: nodecomponentsatellitestandardserializeraw
Description: Serialize the contents of a standard implementation of the satellite component to raw (binary) file.
Code lines: 12
Contained by: module galacticus_nodes

subroutine: nodecomponentsatellitestandardserializevalues
Description: Serialize evolvable properties of a standard implementation of the satellite component to array.
Code lines: 20
Contained by: module galacticus_nodes

subroutine: nodecomponentsatellitestandardserializexml

Description: Serialize the contents of a standard implementation of the satellite component to XML.
Code lines: 19
Contained by: module galacticus_nodes

function: nodecomponentsatellitestandardsizeof
Description: Return the size in bytes of a standard implementation of the satellite component.
Code lines: 10
Contained by: module galacticus_nodes

function: nodecomponentsatellitestandardtype
Description: Returns the type name for the standard implementation of the satellite component class.
Code lines: 13
Contained by: module galacticus_nodes

function: nodecomponentsatellitetype
Description: Returns the type name for the satellite component class.
Code lines: 13
Contained by: module galacticus_nodes

type: nodecomponentsatelliteverysimple
Description: Class for the verySimple implementation of the satellite component.
Code lines: 174
Contained by: module galacticus_nodes

subroutine: nodecomponentsatelliteverysimplebuilder
Description: Build a verySimple implementation of the satellite component from a supplied XML definition.
Code lines: 24
Contained by: module galacticus_nodes
Modules used: fox_dom galacticus_error memory_management

subroutine: nodecomponentsatelliteverysimpledeserializeraw
Description: Deserialize the contents of a verySimple implementation of the satellite component from raw (binary) file.
Code lines: 12
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponentsatelliteverysimpledeserializevalues
Description: Deserialize evolvable properties of a verySimple implementation of the satellite component from array.
Code lines: 16
Contained by: module galacticus_nodes

subroutine: nodecomponentsatelliteverysimplefinalize
Description: Finalize a verySimple implementation of the satellite component.
Code lines: 13
Contained by: module galacticus_nodes

Modules used: memory_management

subroutine: nodecomponentsatelliteverysimpleinitialize
Description: Initialize a verySimple member of the satellite component.
Code lines: 9
Contained by: module galacticus_nodes

function: nodecomponentsatelliteverysimpleisactive
Description: Return true if the verySimple implementation of the satellite component is the active choice.
Code lines: 8
Contained by: module galacticus_nodes

function: nodecomponentsatelliteverysimplenamefromindex
Description: Return the name of the property of given index for a verySimple implementation of the satellite component class.
Code lines: 24
Contained by: module galacticus_nodes
Modules used: iso_varying_string

subroutine: nodecomponentsatelliteverysimpleserializeascii
Description: Serialize the contents of a verySimple implementation of the satellite component to ASCII.
Code lines: 19
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string string_handling

function: nodecomponentsatelliteverysimpleserializecount
Description: Return a count of the serialization of a verySimple implementation of the satellite component.
Code lines: 22
Contained by: module galacticus_nodes

subroutine: nodecomponentsatelliteverysimpleserializeoffsets
Description: Compute offsets into serialization arrays for a verySimple implementation of the satellite component.
Code lines: 28
Contained by: module galacticus_nodes

subroutine: nodecomponentsatelliteverysimpleserializeraw
Description: Serialize the contents of a verySimple implementation of the satellite component to raw (binary) file.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: nodecomponentsatelliteverysimpleserializevalues
Description: Serialize evolvable properties of a verySimple implementation of the satellite component to array.
Code lines: 16
Contained by: module galacticus_nodes

subroutine: `nodecomponentsatelliteverysimpleserializexml`
Description: Serialize the contents of a verySimple implementation of the satellite component to XML.
Code lines: 16
Contained by: module galacticus_nodes

function: `nodecomponentsatelliteverysimplesizeof`
Description: Return the size in bytes of a verySimple implementation of the satellite component.
Code lines: 9
Contained by: module galacticus_nodes

function: `nodecomponentsatelliteverysimpletype`
Description: Returns the type name for the verySimple implementation of the satellite component class.
Code lines: 13
Contained by: module galacticus_nodes

type: `nodecomponentspheroid`
Description: Type for the `spheroid` component class.
Code lines: 971
Contained by: module galacticus_nodes

subroutine: `nodecomponentspheroidbuilder`
Description: Build a generic `spheroid` component from a supplied XML definition.
Code lines: 19
Contained by: module galacticus_nodes
Modules used: `fox_dom` galacticus_error

subroutine: `nodecomponentspheroidcreate`
Description: Create the `spheroid` component of `self`.
Code lines: 38
Contained by: module galacticus_nodes
Modules used: galacticus_display galacticus_error iso_varying_string string_handling

subroutine: `nodecomponentspheroiddestroy`
Description: Destroy the `spheroid` component of `self`
Code lines: 15
Contained by: module galacticus_nodes

subroutine: `nodecomponentspheroiddumpascii`
Description: Dump the content of a `spheroid` component.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string

subroutine: `nodecomponentspheroidfinalize`
Description: Finalize a generic `spheroid` component.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: `nodecomponentspheroidinitialize`
Description: Initialize a generic `spheroid` component.
Code lines: 15
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `nodecomponentspheroidmove`
Description: Move instances of the `spheroid` component, from one node to another.
Code lines: 84
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

type: `nodecomponentspheroidnull`
Description: Class for the null implementation of the spheroid component.
Code lines: 124
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentspheroidnullbuilder`
Description: Build a `null` implementation of the `spheroid` component from a supplied XML definition.
Code lines: 18
Contained by: module `galacticus_nodes`
Modules used: `fox_dom` `galacticus_error` `memory_management`

subroutine: `nodecomponentspheroidnulldeserializeraw`
Description: Deserialize the contents of a null implementation of the spheroid component from raw (binary) file.
Code lines: 16
Contained by: module `galacticus_nodes`
Modules used: `memory_management`

subroutine: `nodecomponentspheroidnulldeserializevalues`
Description: Deserialize evolvable properties of a null implementation of the spheroid component from array.
Code lines: 17
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentspheroidnullfinalize`
Description: Finalize a `null` implementation of the `spheroid` component.
Code lines: 13
Contained by: module `galacticus_nodes`
Modules used: `memory_management`

subroutine: `nodecomponentspheroidnullinitialize`
Description: Initialize a `null` member of the `spheroid` component.
Code lines: 11
Contained by: module `galacticus_nodes`

function: `nodecomponentspheroidnullisactive`
Description: Return true if the null implementation of the spheroid component is the active choice.
Code lines: 8
Contained by: module galacticus_nodes

function: `nodecomponentspheroidnullnamefromindex`
Description: Return the name of the property of given index for a `null` implementation of the `spheroid` component class.
Code lines: 21
Contained by: module galacticus_nodes
Modules used: iso_varying_string

subroutine: `nodecomponentspheroidnullserializeascii`
Description: Serialize the contents of a null implementation of the spheroid component to ASCII.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string string_handling

function: `nodecomponentspheroidnullserializecount`
Description: Return a count of the serialization of a null implementation of the spheroid component.
Code lines: 15
Contained by: module galacticus_nodes

subroutine: `nodecomponentspheroidnullserializeoffsets`
Description: Compute offsets into serialization arrays for a `null` implementation of the `spheroid` component.
Code lines: 22
Contained by: module galacticus_nodes

subroutine: `nodecomponentspheroidnullserializeraw`
Description: Serialize the contents of a null implementation of the spheroid component to raw (binary) file.
Code lines: 14
Contained by: module galacticus_nodes

subroutine: `nodecomponentspheroidnullserializevalues`
Description: Serialize evolvable properties of a `null` implementation of the `spheroid` component to array.
Code lines: 17
Contained by: module galacticus_nodes

subroutine: `nodecomponentspheroidnullserializexml`
Description: Serialize the contents of a null implementation of the spheroid component to XML.
Code lines: 14
Contained by: module galacticus_nodes

function: `nodecomponentspheroidnullsizeof`
Description: Return the size in bytes of a null implementation of the spheroid component.

Code lines: 9
Contained by: module galacticus_nodes

function: nodecomponentspheroidnulltype
Description: Returns the type name for the null implementation of the spheroid component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: nodecomponentspheroidremove
Description: Remove an instance of the spheroid component from a node.
Code lines: 50
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: nodecomponentspheroidserializeascii
Description: Serialize the content of a spheroid component to ASCII.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string

function: nodecomponentspheroidsizeof
Description: Return the size in bytes of a nodeComponentSpheroid component.
Code lines: 9
Contained by: module galacticus_nodes

type: nodecomponentspheroidstandard
Description: Class for the standard implementation of the spheroid component.
Code lines: 777
Contained by: module galacticus_nodes

subroutine: nodecomponentspheroidstandardbuilder
Description: Build a standard implementation of the spheroid component from a supplied XML definition.
Code lines: 90
Contained by: module galacticus_nodes
Modules used: fox_dom galacticus_error memory_management

subroutine: nodecomponentspheroidstandarddeserializeraw
Description: Deserialize the contents of a standard implementation of the spheroid component from raw (binary) file.
Code lines: 23
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponentspheroidstandarddeserializevalues
Description: Deserialize evolvable properties of a standard implementation of the spheroid component from array.
Code lines: 53
Contained by: module galacticus_nodes

subroutine: `nodecomponentspheroidstandardfinalize`
Description: Finalize a `standard` implementation of the `spheroid` component.
Code lines: 15
Contained by: module `galacticus_nodes`
Modules used: `memory_management`

subroutine: `nodecomponentspheroidstandardinitialize`
Description: Initialize a `standard` member of the `spheroid` component.
Code lines: 20
Contained by: module `galacticus_nodes`

function: `nodecomponentspheroidstandardisactive`
Description: Return true if the standard implementation of the spheroid component is the active choice.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `nodecomponentspheroidstandardnamefromindex`
Description: Return the name of the property of given index for a `standard` implementation of the `spheroid` component class.
Code lines: 77
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

subroutine: `nodecomponentspheroidstandardserializeascii`
Description: Serialize the contents of a standard implementation of the spheroid component to ASCII.
Code lines: 57
Contained by: module `galacticus_nodes`
Modules used: `galacticus_display` `iso_varying_string` `string_handling`

function: `nodecomponentspheroidstandardserializecount`
Description: Return a count of the serialization of a standard implementation of the spheroid component.
Code lines: 83
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentspheroidstandardserializeoffsets`
Description: Compute offsets into serialization arrays for a `standard` implementation of the `spheroid` component.
Code lines: 137
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentspheroidstandardserializeraw`
Description: Serialize the contents of a standard implementation of the spheroid component to raw (binary) file.
Code lines: 21
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentspheroidstandardserializevalues`

Description: Serialize evolvable properties of a `standard` implementation of the `spheroid` component to array.
Code lines: 53
Contained by: module galacticus_nodes

subroutine: `nodecomponentspheroidstandardserializexml`
Description: Serialize the contents of a standard implementation of the spheroid component to XML.
Code lines: 33
Contained by: module galacticus_nodes

function: `nodecomponentspheroidstandardsizeof`
Description: Return the size in bytes of a standard implementation of the spheroid component.
Code lines: 14
Contained by: module galacticus_nodes

function: `nodecomponentspheroidstandardtype`
Description: Returns the type name for the standard implementation of the spheroid component class.
Code lines: 13
Contained by: module galacticus_nodes

function: `nodecomponentspheroidtype`
Description: Returns the type name for the spheroid component class.
Code lines: 13
Contained by: module galacticus_nodes

type: `nodecomponentspheroidverysimple`
Description: Class for the verySimple implementation of the spheroid component.
Code lines: 522
Contained by: module galacticus_nodes

subroutine: `nodecomponentspheroidverysimplebuilder`
Description: Build a `verySimple` implementation of the `spheroid` component from a supplied XML definition.
Code lines: 72
Contained by: module galacticus_nodes
Modules used: `fox_dom` galacticus_error memory_management

subroutine: `nodecomponentspheroidverysimpledeserializeraw`
Description: Deserialize the contents of a verySimple implementation of the spheroid component from raw (binary) file.
Code lines: 20
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: `nodecomponentspheroidverysimpledeserializevalues`
Description: Deserialize evolvable properties of a verySimple implementation of the spheroid component from array.
Code lines: 40
Contained by: module galacticus_nodes

subroutine: `nodecomponentspheroidverysimplefinalize`
Description: Finalize a `verySimple` implementation of the `spheroid` component.
Code lines: 14
Contained by: module `galacticus_nodes`
Modules used: `memory_management`

subroutine: `nodecomponentspheroidverysimpleinitialize`
Description: Initialize a `verySimple` member of the `spheroid` component.
Code lines: 17
Contained by: module `galacticus_nodes`

function: `nodecomponentspheroidverysimpleisactive`
Description: Return true if the verySimple implementation of the spheroid component is the active choice.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `nodecomponentspheroidverysimplenamefromindex`
Description: Return the name of the property of given index for a `verySimple` implementation of the `spheroid` component class.
Code lines: 56
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

subroutine: `nodecomponentspheroidverysimpleserializeascii`
Description: Serialize the contents of a verySimple implementation of the spheroid component to ASCII.
Code lines: 47
Contained by: module `galacticus_nodes`
Modules used: `galacticus_display` `iso_varying_string` `string_handling`

function: `nodecomponentspheroidverysimpleserializecount`
Description: Return a count of the serialization of a verySimple implementation of the spheroid component.
Code lines: 59
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentspheroidverysimpleserializeoffsets`
Description: Compute offsets into serialization arrays for a `verySimple` implementation of the `spheroid` component.
Code lines: 95
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentspheroidverysimpleserializeraw`
Description: Serialize the contents of a verySimple implementation of the spheroid component to raw (binary) file.
Code lines: 18
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentspheroidverysimpleserializevalues`

Description: Serialize evolvable properties of a `verySimple` implementation of the `spheroid` component to array.
Code lines: 40
Contained by: module galacticus_nodes

subroutine: `nodecomponentspheroidverysimpleserializexml`
Description: Serialize the contents of a verySimple implementation of the spheroid component to XML.
Code lines: 29
Contained by: module galacticus_nodes

function: `nodecomponentspheroidverysimplesizeof`
Description: Return the size in bytes of a verySimple implementation of the spheroid component.
Code lines: 13
Contained by: module galacticus_nodes

function: `nodecomponentspheroidverysimpletype`
Description: Returns the type name for the verySimple implementation of the spheroid component class.
Code lines: 13
Contained by: module galacticus_nodes

type: `nodecomponentspin`
Description: Type for the `spin` component class.
Code lines: 334
Contained by: module galacticus_nodes

subroutine: `nodecomponentspinbuilder`
Description: Build a generic `spin` component from a supplied XML definition.
Code lines: 19
Contained by: module galacticus_nodes
Modules used: `fox_dom` galacticus_error

subroutine: `nodecomponentspincreate`
Description: Create the `spin` component of `self`.
Code lines: 38
Contained by: module galacticus_nodes
Modules used: galacticus_display galacticus_error iso_varying_string string_handling

subroutine: `nodecomponentspindestroy`
Description: Destroy the `spin` component of `self`
Code lines: 15
Contained by: module galacticus_nodes

subroutine: `nodecomponentspindumpascii`
Description: Dump the content of a `spin` component.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string

subroutine: `nodecomponentspinfinalize`
Description: Finalize a generic `spin` component.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: `nodecomponentspininitialize`
Description: Initialize a generic `spin` component.
Code lines: 15
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: `nodecomponentspinmove`
Description: Move instances of the `spin` component, from one node to another.
Code lines: 112
Contained by: module galacticus_nodes
Modules used: galacticus_error

type: `nodecomponentspinnull`
Description: Class for the null implementation of the spin component.
Code lines: 124
Contained by: module galacticus_nodes

subroutine: `nodecomponentspinnullbuilder`
Description: Build a `null` implementation of the `spin` component from a supplied XML definition.
Code lines: 18
Contained by: module galacticus_nodes
Modules used: fox_dom galacticus_error memory_management

subroutine: `nodecomponentspinnulldeserializeraw`
Description: Deserialize the contents of a null implementation of the spin component from raw (binary) file.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: `nodecomponentspinnulldeserializevalues`
Description: Deserialize evolvable properties of a null implementation of the spin component from array.
Code lines: 17
Contained by: module galacticus_nodes

subroutine: `nodecomponentspinnullfinalize`
Description: Finalize a `null` implementation of the `spin` component.
Code lines: 13
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: `nodecomponentspinnullinitialize`

Description: Initialize a `null` member of the `spin` component.
Code lines: 11
Contained by: module galacticus_nodes

function: `nodecomponentspinnullisactive`
Description: Return true if the null implementation of the spin component is the active choice.
Code lines: 8
Contained by: module galacticus_nodes

function: `nodecomponentspinnullnamefromindex`
Description: Return the name of the property of given index for a `null` implementation of the `spin` component class.
Code lines: 21
Contained by: module galacticus_nodes
Modules used: iso_varying_string

subroutine: `nodecomponentspinnullserializeascii`
Description: Serialize the contents of a null implementation of the spin component to ASCII.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string string_handling

function: `nodecomponentspinnullserializecount`
Description: Return a count of the serialization of a null implementation of the spin component.
Code lines: 15
Contained by: module galacticus_nodes

subroutine: `nodecomponentspinnullserializeoffsets`
Description: Compute offsets into serialization arrays for a `null` implementation of the `spin` component.
Code lines: 22
Contained by: module galacticus_nodes

subroutine: `nodecomponentspinnullserializeraw`
Description: Serialize the contents of a null implementation of the spin component to raw (binary) file.
Code lines: 14
Contained by: module galacticus_nodes

subroutine: `nodecomponentspinnullserializevalues`
Description: Serialize evolvable properties of a `null` implementation of the `spin` component to array.
Code lines: 17
Contained by: module galacticus_nodes

subroutine: `nodecomponentspinnullserializexml`
Description: Serialize the contents of a null implementation of the spin component to XML.
Code lines: 14
Contained by: module galacticus_nodes

function: `nodecomponentspinnullsizeof`

Description: Return the size in bytes of a null implementation of the spin component.
Code lines: 9
Contained by: module galacticus_nodes

function: nodecomponentspinnulltype
Description: Returns the type name for the null implementation of the spin component class.
Code lines: 13
Contained by: module galacticus_nodes

type: nodecomponentspinpreset
Description: Class for the preset implementation of the spin component.
Code lines: 194
Contained by: module galacticus_nodes

type: nodecomponentspinpreset3d
Description: Class for the preset3D implementation of the spin component.
Code lines: 201
Contained by: module galacticus_nodes

subroutine: nodecomponentspinpreset3dbuilder
Description: Build a preset3D implementation of the spin component from a supplied XML definition.
Code lines: 36
Contained by: module galacticus_nodes
Modules used: fox_dom galacticus_error memory_management

subroutine: nodecomponentspinpreset3ddeserializeraw
Description: Deserialize the contents of a preset3D implementation of the spin component from raw (binary) file.
Code lines: 26
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponentspinpreset3ddeserializevalues
Description: Deserialize evolvable properties of a preset3D implementation of the spin component from array.
Code lines: 24
Contained by: module galacticus_nodes

subroutine: nodecomponentspinpreset3dfinalize
Description: Finalize a preset3D implementation of the spin component.
Code lines: 12
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponentspinpreset3dinitialize
Description: Initialize a preset3D member of the spin component.
Code lines: 13
Contained by: module galacticus_nodes

function: `nodecomponentspinpreset3disactive`
Description: Return true if the preset3D implementation of the spin component is the active choice.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `nodecomponentspinpreset3dnamefromindex`
Description: Return the name of the property of given index for a `preset3D` implementation of the `spin` component class.
Code lines: 23
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

subroutine: `nodecomponentspinpreset3dserializeascii`
Description: Serialize the contents of a preset3D implementation of the spin component to ASCII.
Code lines: 32
Contained by: module `galacticus_nodes`
Modules used: `galacticus_display` `iso_varying_string` `string_handling`

function: `nodecomponentspinpreset3dserializecount`
Description: Return a count of the serialization of a preset3D implementation of the spin component.
Code lines: 19
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentspinpreset3dserializeoffsets`
Description: Compute offsets into serialization arrays for a `preset3D` implementation of the `spin` component.
Code lines: 26
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentspinpreset3dserializeraw`
Description: Serialize the contents of a preset3D implementation of the spin component to raw (binary) file.
Code lines: 20
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentspinpreset3dserializevalues`
Description: Serialize evolvable properties of a `preset3D` implementation of the `spin` component to array.
Code lines: 24
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentspinpreset3dserializexml`
Description: Serialize the contents of a preset3D implementation of the spin component to XML.
Code lines: 22
Contained by: module `galacticus_nodes`

function: `nodecomponentspinpreset3dsizeof`
Description: Return the size in bytes of a preset3D implementation of the spin component.

Code lines: 11
Contained by: module galacticus_nodes

function: nodecomponentspinpreset3dtype
Description: Returns the type name for the preset3D implementation of the spin component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: nodecomponentspinpresetbuilder
Description: Build a preset implementation of the spin component from a supplied XML definition.
Code lines: 30
Contained by: module galacticus_nodes
Modules used: fox_dom galacticus_error memory_management

subroutine: nodecomponentspinpresetdeserializeraw
Description: Deserialize the contents of a preset implementation of the spin component from raw (binary) file.
Code lines: 13
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponentspinpresetdeserializevalues
Description: Deserialize evolvable properties of a preset implementation of the spin component from array.
Code lines: 16
Contained by: module galacticus_nodes

subroutine: nodecomponentspinpresetfinalize
Description: Finalize a preset implementation of the spin component.
Code lines: 13
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponentspinpresetinitialize
Description: Initialize a preset member of the spin component.
Code lines: 10
Contained by: module galacticus_nodes

function: nodecomponentspinpresetisactive
Description: Return true if the preset implementation of the spin component is the active choice.
Code lines: 8
Contained by: module galacticus_nodes

function: nodecomponentspinpresetnamefromindex
Description: Return the name of the property of given index for a preset implementation of the spin component class.
Code lines: 24
Contained by: module galacticus_nodes
Modules used: iso_varying_string

subroutine: `nodecomponentspinpresetserializeascii`
Description: Serialize the contents of a preset implementation of the spin component to ASCII.
Code lines: 22
Contained by: module `galacticus_nodes`
Modules used: `galacticus_display` `iso_varying_string` `string_handling`

function: `nodecomponentspinpresetserializecount`
Description: Return a count of the serialization of a preset implementation of the spin component.
Code lines: 22
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentspinpresetserializeoffsets`
Description: Compute offsets into serialization arrays for a `preset` implementation of the `spin` component.
Code lines: 28
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentspinpresetserializeraw`
Description: Serialize the contents of a preset implementation of the spin component to raw (binary) file.
Code lines: 11
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentspinpresetserializevalues`
Description: Serialize evolvable properties of a `preset` implementation of the `spin` component to array.
Code lines: 16
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentspinpresetserializexml`
Description: Serialize the contents of a preset implementation of the spin component to XML.
Code lines: 16
Contained by: module `galacticus_nodes`

function: `nodecomponentspinpresetsizeof`
Description: Return the size in bytes of a preset implementation of the spin component.
Code lines: 9
Contained by: module `galacticus_nodes`

function: `nodecomponentspinpresettype`
Description: Returns the type name for the preset implementation of the spin component class.
Code lines: 13
Contained by: module `galacticus_nodes`

type: `nodecomponentspinrandom`
Description: Class for the random implementation of the spin component.
Code lines: 180
Contained by: module `galacticus_nodes`

subroutine: `nodecomponentspinrandombuilder`
Description: Build a `random` implementation of the `spin` component from a supplied XML definition.
Code lines: 24
Contained by: module galacticus_nodes
Modules used: `fox_dom` galacticus_error memory_management

subroutine: `nodecomponentspinrandomdeserializeraw`
Description: Deserialize the contents of a random implementation of the spin component from raw (binary) file.
Code lines: 12
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: `nodecomponentspinrandomdeserializevalues`
Description: Deserialize evolvable properties of a random implementation of the spin component from array.
Code lines: 16
Contained by: module galacticus_nodes

subroutine: `nodecomponentspinrandomfinalize`
Description: Finalize a `random` implementation of the `spin` component.
Code lines: 13
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: `nodecomponentspinrandominitialize`
Description: Initialize a `random` member of the `spin` component.
Code lines: 9
Contained by: module galacticus_nodes

function: `nodecomponentspinrandomisactive`
Description: Return true if the random implementation of the spin component is the active choice.
Code lines: 8
Contained by: module galacticus_nodes

function: `nodecomponentspinrandomnamefromindex`
Description: Return the name of the property of given index for a `random` implementation of the `spin` component class.
Code lines: 24
Contained by: module galacticus_nodes
Modules used: iso_varying_string

subroutine: `nodecomponentspinrandomserializeascii`
Description: Serialize the contents of a random implementation of the spin component to ASCII.
Code lines: 19
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string

string_handling

function: `nodecomponentspinrandomserializecount`
Description: Return a count of the serialization of a random implementation of the spin component.
Code lines: 22
Contained by: module galacticus_nodes

subroutine: `nodecomponentspinrandomserializeoffsets`
Description: Compute offsets into serialization arrays for a `random` implementation of the `spin` component.
Code lines: 28
Contained by: module galacticus_nodes

subroutine: `nodecomponentspinrandomserializeraw`
Description: Serialize the contents of a random implementation of the spin component to raw (binary) file.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: `nodecomponentspinrandomserializevalues`
Description: Serialize evolvable properties of a `random` implementation of the `spin` component to array.
Code lines: 16
Contained by: module galacticus_nodes

subroutine: `nodecomponentspinrandomserializexml`
Description: Serialize the contents of a random implementation of the spin component to XML.
Code lines: 16
Contained by: module galacticus_nodes

function: `nodecomponentspinrandomsizeof`
Description: Return the size in bytes of a random implementation of the spin component.
Code lines: 9
Contained by: module galacticus_nodes

function: `nodecomponentspinrandomtype`
Description: Returns the type name for the random implementation of the spin component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: `nodecomponentspinremove`
Description: Remove an instance of the `spin` component from a node.
Code lines: 66
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: `nodecomponentspinserializeascii`
Description: Serialize the content of a `spin` component to ASCII.
Code lines: 16
Contained by: module galacticus_nodes

Modules used: galacticus_display iso_varying_string

function: nodecomponentspinsizeof
Description: Return the size in bytes of a nodeComponentSpin component.
Code lines: 9
Contained by: module galacticus_nodes

function: nodecomponentspintype
Description: Returns the type name for the spin component class.
Code lines: 13
Contained by: module galacticus_nodes

type: nodecomponentspinvitvitska
Description: Class for the vitvitska implementation of the spin component.
Code lines: 294
Contained by: module galacticus_nodes

subroutine: nodecomponentspinvitvitskabuilder
Description: Build a vitvitska implementation of the spin component from a supplied XML definition.
Code lines: 33
Contained by: module galacticus_nodes
Modules used: fox_dom galacticus_error memory_management

subroutine: nodecomponentspinvitvitskadeserializeraw
Description: Deserialize the contents of a vitvitska implementation of the spin component from raw (binary) file.
Code lines: 20
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponentspinvitvitskadeserializevalues
Description: Deserialize evolvable properties of a vitvitska implementation of the spin component from array.
Code lines: 23
Contained by: module galacticus_nodes

subroutine: nodecomponentspinvitvitskafinalize
Description: Finalize a vitvitska implementation of the spin component.
Code lines: 11
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: nodecomponentspinvitvitskainitialize
Description: Initialize a vitvitska member of the spin component.
Code lines: 11
Contained by: module galacticus_nodes

function: nodecomponentspinvitvitskaisactive

Description: Return true if the vitvitska implementation of the spin component is the active choice.
Code lines: 8
Contained by: module galacticus_nodes

function: nodecomponentspinvitvitskanamefromindex
Description: Return the name of the property of given index for a vitvitska implementation of the spin component class.
Code lines: 28
Contained by: module galacticus_nodes
Modules used: iso_varying_string

subroutine: nodecomponentspinvitvitskaserializeascii
Description: Serialize the contents of a vitvitska implementation of the spin component to ASCII.
Code lines: 27
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string string_handling

function: nodecomponentspinvitvitskaserializecount
Description: Return a count of the serialization of a vitvitska implementation of the spin component.
Code lines: 27
Contained by: module galacticus_nodes

subroutine: nodecomponentspinvitvitskaserializeoffsets
Description: Compute offsets into serialization arrays for a vitvitska implementation of the spin component.
Code lines: 39
Contained by: module galacticus_nodes

subroutine: nodecomponentspinvitvitskaserializeraw
Description: Serialize the contents of a vitvitska implementation of the spin component to raw (binary) file.
Code lines: 15
Contained by: module galacticus_nodes

subroutine: nodecomponentspinvitvitskaserializevalues
Description: Serialize evolvable properties of a vitvitska implementation of the spin component to array.
Code lines: 23
Contained by: module galacticus_nodes

subroutine: nodecomponentspinvitvitskaserializexml
Description: Serialize the contents of a vitvitska implementation of the spin component to XML.
Code lines: 20
Contained by: module galacticus_nodes

function: nodecomponentspinvitvitskasizeof
Description: Return the size in bytes of a vitvitska implementation of the spin component.
Code lines: 10

Contained by: module galacticus_nodes

function: nodecomponentspinvitvitskatype
Description: Returns the type name for the vitvitska implementation of the spin component class.
Code lines: 13
Contained by: module galacticus_nodes

type: nodeevent
Description: Base class for events attached to nodes.
Code lines: 39
Contained by: module galacticus_nodes

type: nodeeventbranchjump
Description: Class for branch jump events attached to nodes.
Code lines: 26
Contained by: module galacticus_nodes

subroutine: nodeeventbranchjumpdeserializeraw
Description: Deserialize a nodeEventBranchJump object from raw file.
Code lines: 13
Contained by: module galacticus_nodes
Modules used: iso_c_binding

type: nodeeventbranchjumpintertree
Description: Class for inter-tree branch jump events attached to nodes.
Code lines: 32
Contained by: module galacticus_nodes

subroutine: nodeeventbranchjumpintertreedeserializeraw
Description: Deserialize a nodeEventBranchJumpInterTree object from raw file.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: iso_c_binding

subroutine: nodeeventbranchjumpintertreeserializeraw
Description: Serialize a nodeEventBranchJumpInterTree object to raw file.
Code lines: 19
Contained by: module galacticus_nodes
Modules used: iso_c_binding

subroutine: nodeeventbranchjumpserializeraw
Description: Serialize a nodeEventBranchJump object to raw file.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: iso_c_binding

function: nodeeventbuildfromraw
Description: Build a nodeEvent class object from a raw dump file.
Code lines: 26

Contained by: module galacticus_nodes

subroutine: nodeeventdeserializeraw
Description: Deserialize a nodeEvent object from raw file.
Code lines: 22
Contained by: module galacticus_nodes
Modules used: iso_c_binding

subroutine: nodeeventserializeraw
Description: Serialize a nodeEvent object to raw file.
Code lines: 24
Contained by: module galacticus_nodes
Modules used: iso_c_binding

function: nodeeventsizeof
Description: Compute the size of the non-static parts of a spin object.
Code lines: 12
Contained by: module galacticus_nodes

type: nodeeventsubhalopromotion
Description: Class for subhalo promotion events attached to nodes.
Code lines: 26
Contained by: module galacticus_nodes

subroutine: nodeeventsubhalopromotiondeserializeraw
Description: Deserialize a nodeEventSubhaloPromotion object from raw file.
Code lines: 13
Contained by: module galacticus_nodes
Modules used: iso_c_binding

type: nodeeventsubhalopromotionintertree
Description: Class for inter-tree subhalo promotion events attached to nodes.
Code lines: 32
Contained by: module galacticus_nodes

subroutine: nodeeventsubhalopromotionintertreedeserializeraw
Description: Deserialize a nodeEventSubhaloPromotionInterTree object from raw file.
Code lines: 16
Contained by: module galacticus_nodes
Modules used: iso_c_binding

subroutine: nodeeventsubhalopromotionintertreeserializeraw
Description: Serialize a nodeEventSubhaloPromotionInterTree object to raw file.
Code lines: 19
Contained by: module galacticus_nodes
Modules used: iso_c_binding

subroutine: nodeeventsubhalopromotionserializeraw
Description: Serialize a nodeEventSubhaloPromotion object to raw file.

Code lines: 16
Contained by: module galacticus_nodes
Modules used: iso_c_binding

function: nullagestatisticsgetininteger0
Description: A null get rate function for a rank 0 nodecomponentagestatistics class.
Code lines: 14
Contained by: module galacticus_nodes

subroutine: nullagestatisticsscaleinoutdouble0
Description: A null scale rate function for a rank 0 nodecomponentagestatistics class.
Code lines: 17
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: nullagestatisticssetinoutdouble0
Description: A null set rate function for a rank 0 nodecomponentagestatistics class.
Code lines: 18
Contained by: module galacticus_nodes
Modules used: galacticus_error

function: nullbasicgetininteger0
Description: A null get rate function for a rank 0 nodecomponentbasic class.
Code lines: 14
Contained by: module galacticus_nodes

subroutine: nullbasicrateinoutdouble0
Description: A null rate rate function for a rank 0 nodecomponentbasic class.
Code lines: 21
Contained by: module galacticus_nodes

subroutine: nullbasicscaleinoutdouble0
Description: A null scale rate function for a rank 0 nodecomponentbasic class.
Code lines: 17
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: nullbasicsetinoutdouble0
Description: A null set rate function for a rank 0 nodecomponentbasic class.
Code lines: 18
Contained by: module galacticus_nodes
Modules used: galacticus_error

function: nullblackholegetininteger0
Description: A null get rate function for a rank 0 nodecomponentblackhole class.
Code lines: 14
Contained by: module galacticus_nodes

subroutine: nullblackholerateinoutdouble0

Description: A null rate rate function for a rank 0 `nodecomponentblackhole` class.
Code lines: 21
Contained by: module galacticus_nodes

subroutine: `nullblackholescaleinoutdouble0`
Description: A null scale rate function for a rank 0 `nodecomponentblackhole` class.
Code lines: 17
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: `nullblackholesetinoutdouble0`
Description: A null set rate function for a rank 0 `nodecomponentblackhole` class.
Code lines: 18
Contained by: module galacticus_nodes
Modules used: galacticus_error

function: `nulldarkmatterprofilegetininteger0`
Description: A null get rate function for a rank 0 `nodecomponentdarkmatterprofile` class.
Code lines: 14
Contained by: module galacticus_nodes

subroutine: `nulldarkmatterprofilerateinoutdouble0`
Description: A null rate rate function for a rank 0 `nodecomponentdarkmatterprofile` class.
Code lines: 21
Contained by: module galacticus_nodes

subroutine: `nulldarkmatterprofilescaleinoutdouble0`
Description: A null scale rate function for a rank 0 `nodecomponentdarkmatterprofile` class.
Code lines: 17
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: `nulldarkmatterprofilesetinoutdouble0`
Description: A null set rate function for a rank 0 `nodecomponentdarkmatterprofile` class.
Code lines: 18
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: `nulldarkmatterprofilesetinoutlogical0`
Description: A null set rate function for a rank 0 `nodecomponentdarkmatterprofile` class.
Code lines: 18
Contained by: module galacticus_nodes
Modules used: galacticus_error

function: `nulldiskgetininteger0`
Description: A null get rate function for a rank 0 `nodecomponentdisk` class.
Code lines: 14
Contained by: module galacticus_nodes

subroutine: `nulldiskrateinoutabundances0`
Description: A null rate rate function for a rank 0 `nodecomponentdisk` class.
Code lines: 21
Contained by: module `galacticus_nodes`

subroutine: `nulldiskrateinoutdouble0`
Description: A null rate rate function for a rank 0 `nodecomponentdisk` class.
Code lines: 21
Contained by: module `galacticus_nodes`

subroutine: `nulldiskrateinouthistory0`
Description: A null rate rate function for a rank 0 `nodecomponentdisk` class.
Code lines: 21
Contained by: module `galacticus_nodes`

subroutine: `nulldiskrateinoutstellarluminosities0`
Description: A null rate rate function for a rank 0 `nodecomponentdisk` class.
Code lines: 21
Contained by: module `galacticus_nodes`

subroutine: `nulldiskscaleinoutabundances0`
Description: A null scale rate function for a rank 0 `nodecomponentdisk` class.
Code lines: 17
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `nulldiskscaleinoutdouble0`
Description: A null scale rate function for a rank 0 `nodecomponentdisk` class.
Code lines: 17
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `nulldiskscaleinouthistory0`
Description: A null scale rate function for a rank 0 `nodecomponentdisk` class.
Code lines: 17
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `nulldiskscaleinoutstellarluminosities0`
Description: A null scale rate function for a rank 0 `nodecomponentdisk` class.
Code lines: 17
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `nulldisksetinoutabundances0`
Description: A null set rate function for a rank 0 `nodecomponentdisk` class.
Code lines: 18
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `nulldisksetinoutdouble0`
Description: A null set rate function for a rank 0 `nodecomponentdisk` class.
Code lines: 18
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `nulldisksetinouthistory0`
Description: A null set rate function for a rank 0 `nodecomponentdisk` class.
Code lines: 18
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `nulldisksetinoutlogical0`
Description: A null set rate function for a rank 0 `nodecomponentdisk` class.
Code lines: 18
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `nulldisksetinoutstellarluminosities0`
Description: A null set rate function for a rank 0 `nodecomponentdisk` class.
Code lines: 18
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `nulldynamicsstatisticssetinoutdouble1`
Description: A null set rate function for a rank 1 `nodecomponentdynamicsstatistics` class.
Code lines: 18
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `nullformationtimesetinoutdouble0`
Description: A null set rate function for a rank 0 `nodecomponentformationtime` class.
Code lines: 18
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `nullgenericrateinoutabundances0`
Description: A null rate rate function for a rank 0 `nodecomponent` class.
Code lines: 21
Contained by: module `galacticus_nodes`

subroutine: `nullgenericrateinoutdouble0`
Description: A null rate rate function for a rank 0 `nodecomponent` class.
Code lines: 21
Contained by: module `galacticus_nodes`

subroutine: `nullhosthistorysetinoutdouble0`
Description: A null set rate function for a rank 0 `nodecomponenthosthistory` class.

Code lines: 18
Contained by: module galacticus_nodes
Modules used: galacticus_error

function: nullhothalogetininteger0
Description: A null get rate function for a rank 0 nodecomponenthothalo class.
Code lines: 14
Contained by: module galacticus_nodes

subroutine: nullhothalorateinoutabundances0
Description: A null rate rate function for a rank 0 nodecomponenthothalo class.
Code lines: 21
Contained by: module galacticus_nodes

subroutine: nullhothalorateinoutdouble0
Description: A null rate rate function for a rank 0 nodecomponenthothalo class.
Code lines: 21
Contained by: module galacticus_nodes

subroutine: nullhothaloscaleinoutabundances0
Description: A null scale rate function for a rank 0 nodecomponenthothalo class.
Code lines: 17
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: nullhothaloscaleinoutchemicalabundances0
Description: A null scale rate function for a rank 0 nodecomponenthothalo class.
Code lines: 17
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: nullhothaloscaleinoutdouble0
Description: A null scale rate function for a rank 0 nodecomponenthothalo class.
Code lines: 17
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: nullhothalosetinoutabundances0
Description: A null set rate function for a rank 0 nodecomponenthothalo class.
Code lines: 18
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: nullhothalosetinoutchemicalabundances0
Description: A null set rate function for a rank 0 nodecomponenthothalo class.
Code lines: 18
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: `nullhothalosetinoutdouble0`
Description: A null set rate function for a rank 0 `nodecomponenthothalo` class.
Code lines: 18
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `nullhothalosetinoutlogical0`
Description: A null set rate function for a rank 0 `nodecomponenthothalo` class.
Code lines: 18
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `nullindicessetinoutlonginteger0`
Description: A null set rate function for a rank 0 `nodecomponentindices` class.
Code lines: 18
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

function: `nullinteroutputgetininteger0`
Description: A null get rate function for a rank 0 `nodecomponentinteroutput` class.
Code lines: 14
Contained by: module `galacticus_nodes`

subroutine: `nullinteroutputscaleinoutdouble0`
Description: A null scale rate function for a rank 0 `nodecomponentinteroutput` class.
Code lines: 17
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `nullinteroutputsetinoutdouble0`
Description: A null set rate function for a rank 0 `nodecomponentinteroutput` class.
Code lines: 18
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

function: `nullmassflowstatisticsgetininteger0`
Description: A null get rate function for a rank 0 `nodecomponentmassflowstatistics` class.
Code lines: 14
Contained by: module `galacticus_nodes`

subroutine: `nullmassflowstatisticsrateinoutdouble0`
Description: A null rate rate function for a rank 0 `nodecomponentmassflowstatistics` class.
Code lines: 21
Contained by: module `galacticus_nodes`

subroutine: `nullmassflowstatisticsscaleinoutdouble0`
Description: A null scale rate function for a rank 0 `nodecomponentmassflowstatistics` class.
Code lines: 17

Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: nullmassflowstatisticssetinoutdouble0
Description: A null set rate function for a rank 0 nodecomponentmassflowstatistics class.
Code lines: 18
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: nullmergingstatisticssetinoutdouble0
Description: A null set rate function for a rank 0 nodecomponentmergingstatistics class.
Code lines: 18
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: nullmergingstatisticssetinoutdouble1
Description: A null set rate function for a rank 1 nodecomponentmergingstatistics class.
Code lines: 18
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: nullmergingstatisticssetinoutinteger0
Description: A null set rate function for a rank 0 nodecomponentmergingstatistics class.
Code lines: 18
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: nullmergingstatisticssetinoutinteger1
Description: A null set rate function for a rank 1 nodecomponentmergingstatistics class.
Code lines: 18
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: nullnbodysetinoutdouble1
Description: A null set rate function for a rank 1 nodecomponentnbody class.
Code lines: 18
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: nullnbodysetinoutlonginteger1
Description: A null set rate function for a rank 1 nodecomponentnbody class.
Code lines: 18
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: nullpositionsetinoutdouble0
Description: A null set rate function for a rank 0 nodecomponentposition class.
Code lines: 18
Contained by: module galacticus_nodes

Modules used: galacticus_error

subroutine: nullpositionsetinoutdouble1
Description: A null set rate function for a rank 1 nodecomponentposition class.
Code lines: 18
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: nullpositionsetinouthistory0
Description: A null set rate function for a rank 0 nodecomponentposition class.
Code lines: 18
Contained by: module galacticus_nodes
Modules used: galacticus_error

function: nullsatellitegetininteger0
Description: A null get rate function for a rank 0 nodecomponentsatellite class.
Code lines: 14
Contained by: module galacticus_nodes

subroutine: nullsatelliterateinoutdouble0
Description: A null rate rate function for a rank 0 nodecomponentsatellite class.
Code lines: 21
Contained by: module galacticus_nodes

subroutine: nullsatelliterateinoutdouble1
Description: A null rate rate function for a rank 1 nodecomponentsatellite class.
Code lines: 21
Contained by: module galacticus_nodes

subroutine: nullsatelliterateinouttensorrank2dimension3symmetric0
Description: A null rate rate function for a rank 0 nodecomponentsatellite class.
Code lines: 21
Contained by: module galacticus_nodes

subroutine: nullsatellitescaleinoutdouble0
Description: A null scale rate function for a rank 0 nodecomponentsatellite class.
Code lines: 17
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: nullsatellitescaleinoutdouble1
Description: A null scale rate function for a rank 1 nodecomponentsatellite class.
Code lines: 17
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: nullsatellitescaleinouttensorrank2dimension3symmetric0
Description: A null scale rate function for a rank 0 nodecomponentsatellite class.
Code lines: 17

Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: nullsatellitesetinoutdouble0
Description: A null set rate function for a rank 0 nodecomponentsatellite class.
Code lines: 18
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: nullsatellitesetinoutdouble1
Description: A null set rate function for a rank 1 nodecomponentsatellite class.
Code lines: 18
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: nullsatellitesetinouthistory0
Description: A null set rate function for a rank 0 nodecomponentsatellite class.
Code lines: 18
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: nullsatellitesetinoutkeplerorbit0
Description: A null set rate function for a rank 0 nodecomponentsatellite class.
Code lines: 18
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: nullsatellitesetinoutlogical0
Description: A null set rate function for a rank 0 nodecomponentsatellite class.
Code lines: 18
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: nullsatellitesetinoutlongintegerhistory0
Description: A null set rate function for a rank 0 nodecomponentsatellite class.
Code lines: 18
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: nullsatellitesetinouttensorrank2dimension3symmetric0
Description: A null set rate function for a rank 0 nodecomponentsatellite class.
Code lines: 18
Contained by: module galacticus_nodes
Modules used: galacticus_error

function: nullspheroidgetininteger0
Description: A null get rate function for a rank 0 nodecomponentspheroid class.
Code lines: 14
Contained by: module galacticus_nodes

subroutine: `nullspheroidrateinoutdouble0`
Description: A null rate rate function for a rank 0 `nodecomponentspheroid` class.
Code lines: 21
Contained by: module `galacticus_nodes`

subroutine: `nullspheroidrateinouthistory0`
Description: A null rate rate function for a rank 0 `nodecomponentspheroid` class.
Code lines: 21
Contained by: module `galacticus_nodes`

subroutine: `nullspheroidscaleinoutabundances0`
Description: A null scale rate function for a rank 0 `nodecomponentspheroid` class.
Code lines: 17
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `nullspheroidscaleinoutdouble0`
Description: A null scale rate function for a rank 0 `nodecomponentspheroid` class.
Code lines: 17
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `nullspheroidscaleinouthistory0`
Description: A null scale rate function for a rank 0 `nodecomponentspheroid` class.
Code lines: 17
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `nullspheroidscaleinoutstellarluminosities0`
Description: A null scale rate function for a rank 0 `nodecomponentspheroid` class.
Code lines: 17
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `nullspheroidsetinoutabundances0`
Description: A null set rate function for a rank 0 `nodecomponentspheroid` class.
Code lines: 18
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `nullspheroidsetinoutdouble0`
Description: A null set rate function for a rank 0 `nodecomponentspheroid` class.
Code lines: 18
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `nullspheroidsetinouthistory0`

Description: A null set rate function for a rank 0 `nodecomponentspheroid` class.
Code lines: 18
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: `nullspheroidsetinoutlogical0`
Description: A null set rate function for a rank 0 `nodecomponentspheroid` class.
Code lines: 18
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: `nullspheroidsetinoutstellarluminosities0`
Description: A null set rate function for a rank 0 `nodecomponentspheroid` class.
Code lines: 18
Contained by: module galacticus_nodes
Modules used: galacticus_error

function: `nullspingetininteger0`
Description: A null get rate function for a rank 0 `nodecomponentspin` class.
Code lines: 14
Contained by: module galacticus_nodes

subroutine: `nullspinrateinoutdouble0`
Description: A null rate rate function for a rank 0 `nodecomponentspin` class.
Code lines: 21
Contained by: module galacticus_nodes

subroutine: `nullspinrateinoutdouble1`
Description: A null rate rate function for a rank 1 `nodecomponentspin` class.
Code lines: 21
Contained by: module galacticus_nodes

subroutine: `nullspinscaleinoutdouble0`
Description: A null scale rate function for a rank 0 `nodecomponentspin` class.
Code lines: 17
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: `nullspinscaleinoutdouble1`
Description: A null scale rate function for a rank 1 `nodecomponentspin` class.
Code lines: 17
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: `nullspinsetinoutdouble0`
Description: A null set rate function for a rank 0 `nodecomponentspin` class.
Code lines: 18
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: `nullspinsetinoutdouble1`
Description: A null set rate function for a rank 1 `nodecomponentspin` class.
Code lines: 18
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `positionnulloutputcount`
Description: Increment the count of properties to output for a `null` implementation of the `position` component.
Code lines: 22
Contained by: module `galacticus_nodes`

subroutine: `positionnulloutputnames`
Description: Return the names of properties to output for a `null` implementation of the `position` component.
Code lines: 39
Contained by: module `galacticus_nodes`

subroutine: `positionoutput`
Description: Populate output buffers with properties to output for a `position` component.
Code lines: 42
Contained by: module `galacticus_nodes`
Modules used: `multi_counters`

subroutine: `positionoutputcount`
Description: Increment the count of properties to output for a generic `position` component.
Code lines: 15
Contained by: module `galacticus_nodes`

subroutine: `positionoutputnames`
Description: Establish the names of properties to output for a generic `position` component.
Code lines: 20
Contained by: module `galacticus_nodes`

function: `positionposition`
Description: Returns the default value for the `position` property for the `position` component class.
Code lines: 28
Contained by: module `galacticus_nodes`

function: `positionpositionattributematch`
Description: Return a text list of component implementations in the `position` class that have the desired attributes for the `position` property
Code lines: 52
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `positionpositionhistory`
Description: Returns the default value for the `positionHistory` property for the `position` component class.

Code lines: 13
Contained by: module galacticus_nodes

function: positionpositionhistoryattributematch
Description: Return a text list of component implementations in the position class that have the desired attributes for the positionHistory property
Code lines: 41
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: positionpositionhistoryisgettable
Description: Returns true if the positionHistory property is gettable for the position component class.
Code lines: 8
Contained by: module galacticus_nodes

function: positionpositionhistoryrateget
Description: Returns a zero rate for the positionHistory property for the position component class.
Code lines: 13
Contained by: module galacticus_nodes

function: positionpositionisgettable
Description: Returns true if the position property is gettable for the position component class.
Code lines: 11
Contained by: module galacticus_nodes

function: positionpositionorphan
Description: Returns the default value for the positionOrphan property for the position component class.
Code lines: 13
Contained by: module galacticus_nodes

function: positionpositionorphanattributematch
Description: Return a text list of component implementations in the position class that have the desired attributes for the positionOrphan property
Code lines: 30
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: positionpositionorphanisgettable
Description: Returns true if the positionOrphan property is gettable for the position component class.
Code lines: 8
Contained by: module galacticus_nodes

function: positionpositionorphanrateget
Description: Returns a zero rate for the positionOrphan property for the position component class.
Code lines: 13
Contained by: module galacticus_nodes

function: positionpositionrateget

Description: Returns a zero rate for the `position` property for the `position` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

subroutine: `positionpostoutput`
Description: Perform post-output processing of a `position` component.
Code lines: 14
Contained by: module `galacticus_nodes`

subroutine: `positionpresetorphansoutputcount`
Description: Increment the count of properties to output for a `presetOrphans` implementation of the `position` component.
Code lines: 12
Contained by: module `galacticus_nodes`

subroutine: `positionpresetorphansoutputnames`
Description: Return the names of properties to output for a `presetOrphans` implementation of the `position` component.
Code lines: 17
Contained by: module `galacticus_nodes`

function: `positionpresetorphansposition`
Description: Return the position of the node.
Code lines: 35
Contained by: module `galacticus_nodes`

function: `positionpresetorphanspositionorphanget`
Description: Get the value of the `positionOrphan` property of the `presetOrphans` implementation of the `position` component using a deferred function.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `positionpresetorphanspositionorphangetfunction`
Description: Set the function to be used for the `get` method of the `positionOrphan` property of the `PositionPresetOrphans` component.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `positionpresetorphanspositionorphangetisattached`
Description: Return true if the deferred function used to get the `positionOrphan` property of the `PositionPresetOrphans` component class has been attached.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `positionpresetorphanspositionorphangetvalue`
Description: Get the `positionOrphan` property of an `presetOrphans` implementation of the `position` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `positionpresetorphanspositionorphanset`
Description: Set the `positionOrphan` property of an `presetOrphans` implementation of the `position` component class.
Code lines: 20
Contained by: module `galacticus_nodes`
Modules used: `memory_management`

subroutine: `positionpresetorphanspositionset`
Description: Set the `position` property of an `presetOrphans` implementation of the `position` component class.
Code lines: 20
Contained by: module `galacticus_nodes`
Modules used: `memory_management`

subroutine: `positionpresetorphanspostoutput`
Description: Perform post-output processing for a `presetOrphans` implementation of the `position` component.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `positionpresetorphanstimeassignget`
Description: Get the `timeAssign` property of an `presetOrphans` implementation of the `position` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `positionpresetorphanstimeassignset`
Description: Set the `timeAssign` property of an `presetOrphans` implementation of the `position` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `positionpresetorphansvelocity`
Description: Return the velocity of the node.
Code lines: 35
Contained by: module `galacticus_nodes`

function: `positionpresetorphansvelocityorphanget`
Description: Get the value of the `velocityOrphan` property of the `presetOrphans` implementation of the `position` component using a deferred function.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `positionpresetorphansvelocityorphangetfunction`
Description: Set the function to be used for the `get` method of the `velocityOrphan` property of the `PositionPresetOrphans` component.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `positionpresetorphansvelocityorphangetisattached`
Description: Return true if the deferred function used to get the `velocityOrphan` property of the `PositionPresetOrphans` component class has been attached.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `positionpresetorphansvelocityorphangetvalue`
Description: Get the `velocityOrphan` property of an `presetOrphans` implementation of the `position` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `positionpresetorphansvelocityorphanset`
Description: Set the `velocityOrphan` property of an `presetOrphans` implementation of the `position` component class.
Code lines: 20
Contained by: module `galacticus_nodes`
Modules used: `memory_management`

subroutine: `positionpresetorphansvelocityset`
Description: Set the `velocity` property of an `presetOrphans` implementation of the `position` component class.
Code lines: 20
Contained by: module `galacticus_nodes`
Modules used: `memory_management`

subroutine: `positionpresetoutputcount`
Description: Increment the count of properties to output for a `preset` implementation of the `position` component.
Code lines: 21
Contained by: module `galacticus_nodes`

subroutine: `positionpresetoutputnames`
Description: Return the names of properties to output for a `preset` implementation of the `position` component.
Code lines: 59
Contained by: module `galacticus_nodes`

function: `positionpresetposition`
Description: Return the position of the node.
Code lines: 36
Contained by: module `galacticus_nodes`
Modules used: `fgsl` `iso_c_binding` `numerical_interpolation`

function: `positionpresetpositionhistoryget`
Description: Get the `positionHistory` property of an `preset` implementation of the `position` component class.
Code lines: 10

Contained by: module galacticus_nodes

subroutine: positionpresetpositionhistoryset
Description: Set the positionHistory property of an preset implementation of the position component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: positionpresetpositionset
Description: Set the position property of an preset implementation of the position component class.
Code lines: 20
Contained by: module galacticus_nodes
Modules used: memory_management

function: positionpresetvelocity
Description: Return the velocity of the node.
Code lines: 36
Contained by: module galacticus_nodes
Modules used: fgsl iso_c_binding numerical_interpolation

subroutine: positionpresetvelocityset
Description: Set the velocity property of an preset implementation of the position component class.
Code lines: 20
Contained by: module galacticus_nodes
Modules used: memory_management

function: positiontimeassign
Description: Returns the default value for the timeAssign property for the position component class.
Code lines: 17
Contained by: module galacticus_nodes

function: positiontimeassignattributematch
Description: Return a text list of component implementations in the position class that have the desired attributes for the timeAssign property
Code lines: 30
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: positiontimeassignisgettable
Description: Returns true if the timeAssign property is gettable for the position component class.
Code lines: 9
Contained by: module galacticus_nodes

function: positiontimeassignrateget
Description: Returns a zero rate for the timeAssign property for the position component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: `positiontracedarkmatteroutputcount`
Description: Increment the count of properties to output for a `traceDarkMatter` implementation of the `position` component.
Code lines: 21
Contained by: module galacticus_nodes

subroutine: `positiontracedarkmatteroutputnames`
Description: Return the names of properties to output for a `traceDarkMatter` implementation of the `position` component.
Code lines: 45
Contained by: module galacticus_nodes

function: `positiontracedarkmatterpositionget`
Description: Get the `position` property of an `traceDarkMatter` implementation of the `position` component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: `positiontracedarkmatterpositionset`
Description: Set the `position` property of an `traceDarkMatter` implementation of the `position` component class.
Code lines: 20
Contained by: module galacticus_nodes
Modules used: memory_management

function: `positionvelocity`
Description: Returns the default value for the `velocity` property for the `position` component class.
Code lines: 23
Contained by: module galacticus_nodes

function: `positionvelocityattributematch`
Description: Return a text list of component implementations in the `position` class that have the desired attributes for the `velocity` property
Code lines: 41
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: `positionvelocityisgettable`
Description: Returns true if the `velocity` property is gettable for the `position` component class.
Code lines: 10
Contained by: module galacticus_nodes

function: `positionvelocityorphan`
Description: Returns the default value for the `velocityOrphan` property for the `position` component class.
Code lines: 13
Contained by: module galacticus_nodes

function: `positionvelocityorphanattributematch`

Description: Return a text list of component implementations in the `position` class that have the desired attributes for the `velocityOrphan` property
Code lines: 30
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `positionvelocityorphanisgettable`
Description: Returns true if the `velocityOrphan` property is gettable for the `position` component class.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `positionvelocityorphanrateget`
Description: Returns a zero rate for the `velocityOrphan` property for the `position` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `positionvelocityrateget`
Description: Returns a zero rate for the `velocity` property for the `position` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `satelliteboundmass`
Description: Returns the default value for the `boundMass` property for the `satellite` component class.
Code lines: 33
Contained by: module `galacticus_nodes`

function: `satelliteboundmassattributematch`
Description: Return a text list of component implementations in the `satellite` class that have the desired attributes for the `boundMass` property
Code lines: 48
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `satelliteboundmasshistory`
Description: Returns the default value for the `boundMassHistory` property for the `satellite` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `satelliteboundmasshistoryattributematch`
Description: Return a text list of component implementations in the `satellite` class that have the desired attributes for the `boundMassHistory` property
Code lines: 30
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `satelliteboundmasshistoryisgettable`
Description: Returns true if the `boundMassHistory` property is gettable for the `satellite` component class.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `satelliteboundmasshistoryrateget`
Description: Returns a zero rate for the `boundMassHistory` property for the `satellite` component class.
Code lines: 13
Contained by: module galacticus_nodes

function: `satelliteboundmassisgettable`
Description: Returns true if the `boundMass` property is gettable for the `satellite` component class.
Code lines: 11
Contained by: module galacticus_nodes

function: `satelliteboundmassrateget`
Description: Returns a zero rate for the `boundMass` property for the `satellite` component class.
Code lines: 13
Contained by: module galacticus_nodes

function: `satelliteisorphan`
Description: Returns the default value for the `isOrphan` property for the `satellite` component class.
Code lines: 13
Contained by: module galacticus_nodes

function: `satelliteisorphanattributematch`
Description: Return a text list of component implementations in the `satellite` class that have the desired attributes for the `isOrphan` property
Code lines: 30
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: `satelliteisorphanisgettable`
Description: Returns true if the `isOrphan` property is gettable for the `satellite` component class.
Code lines: 8
Contained by: module galacticus_nodes

function: `satelliteisorphanrateget`
Description: Returns a zero rate for the `isOrphan` property for the `satellite` component class.
Code lines: 13
Contained by: module galacticus_nodes

function: `satellitemergetime`
Description: Returns the default value for the `mergeTime` property for the `satellite` component class.
Code lines: 29
Contained by: module galacticus_nodes

function: `satellitemergetimeattributematch`
Description: Return a text list of component implementations in the `satellite` class that have the desired attributes for the `mergeTime` property
Code lines: 59
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: `satellitemergetimeisgettable`
Description: Returns true if the `mergeTime` property is gettable for the `satellite` component class.
Code lines: 12
Contained by: module `galacticus_nodes`

function: `satellitemergetimerateget`
Description: Returns a zero rate for the `mergeTime` property for the `satellite` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `satellitenodeindex`
Description: Returns the default value for the `nodeIndex` property for the `satellite` component class.
Code lines: 17
Contained by: module `galacticus_nodes`

function: `satellitenodeindexattributematch`
Description: Return a text list of component implementations in the `satellite` class that have the desired attributes for the `nodeIndex` property
Code lines: 30
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `satellitenodeindexhistory`
Description: Returns the default value for the `nodeIndexHistory` property for the `satellite` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `satellitenodeindexhistoryattributematch`
Description: Return a text list of component implementations in the `satellite` class that have the desired attributes for the `nodeIndexHistory` property
Code lines: 30
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `satellitenodeindexhistoryisgettable`
Description: Returns true if the `nodeIndexHistory` property is gettable for the `satellite` component class.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `satellitenodeindexhistoryrateget`
Description: Returns a zero rate for the `nodeIndexHistory` property for the `satellite` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `satellitenodeindexisgettable`
Description: Returns true if the `nodeIndex` property is gettable for the `satellite` component class.
Code lines: 9

Contained by: module galacticus_nodes

function: satellitenodeindexrateget
Description: Returns a zero rate for the nodeIndex property for the satellite component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: satellitenulloutputcount
Description: Increment the count of properties to output for a null implementation of the satellite component.
Code lines: 22
Contained by: module galacticus_nodes

subroutine: satellitenulloutputnames
Description: Return the names of properties to output for a null implementation of the satellite component.
Code lines: 39
Contained by: module galacticus_nodes

function: satelliteorbitingboundmasscount
Description: Return a count of the number of scalar properties in the boundMass property of an orbiting implementation of the satellite component class.
Code lines: 12
Contained by: module galacticus_nodes

function: satelliteorbitingboundmassget
Description: Get the boundMass property of an orbiting implementation of the satellite component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: satelliteorbitingboundmassjcbnzr
Description: Indicate that the boundMass property of an orbiting implementation of the satellite component class is inactive for differential equation solving.
Code lines: 12
Contained by: module galacticus_nodes

subroutine: satelliteorbitingboundmassrate
Description: Accumulate to the rate of change of the boundMass property of an orbiting implementation of the satellite component class.
Code lines: 31
Contained by: module galacticus_nodes

function: satelliteorbitingboundmassrateget
Description: Get the rate of change of the boundMass property of an orbiting implementation of the satellite component class.
Code lines: 23
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: `satelliteorbitingboundmassscale`
Description: Set the absolute scale of the `boundMass` property of an `orbiting` implementation of the `satellite` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

subroutine: `satelliteorbitingboundmassset`
Description: Set the `boundMass` property of an `orbiting` implementation of the `satellite` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `satelliteorbitingmergetimeget`
Description: Get the `mergeTime` property of an `orbiting` implementation of the `satellite` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `satelliteorbitingmergetimeset`
Description: Set the `mergeTime` property of an `orbiting` implementation of the `satellite` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `satelliteorbitingoutputcount`
Description: Increment the count of properties to output for a `orbiting` implementation of the `satellite` component.
Code lines: 21
Contained by: module `galacticus_nodes`

subroutine: `satelliteorbitingoutputnames`
Description: Return the names of properties to output for a `orbiting` implementation of the `satellite` component.
Code lines: 71
Contained by: module `galacticus_nodes`

function: `satelliteorbitingpositioncount`
Description: Return a count of the number of scalar properties in the `position` property of an `orbiting` implementation of the `satellite` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `satelliteorbitingpositionget`
Description: Get the `position` property of an `orbiting` implementation of the `satellite` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `satelliteorbitingpositionjcbnzr`

Description: Indicate that the `position` property of an `orbiting` implementation of the `satellite` component class is inactive for differential equation solving.
Code lines: 12
Contained by: module `galacticus_nodes`

subroutine: `satelliteorbitingpositionrate`
Description: Accumulate to the rate of change of the `position` property of an `orbiting` implementation of the `satellite` component class.
Code lines: 33
Contained by: module `galacticus_nodes`

function: `satelliteorbitingpositionrateget`
Description: Get the rate of change of the `position` property of an `orbiting` implementation of the `satellite` component class.
Code lines: 25
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `satelliteorbitingpositionscale`
Description: Set the absolute scale of the `position` property of an `orbiting` implementation of the `satellite` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

subroutine: `satelliteorbitingpositionset`
Description: Set the `position` property of an `orbiting` implementation of the `satellite` component class.
Code lines: 20
Contained by: module `galacticus_nodes`
Modules used: `memory_management`

function: `satelliteorbitingtidalheatingnormalizedcount`
Description: Return a count of the number of scalar properties in the `tidalHeatingNormalized` property of an `orbiting` implementation of the `satellite` component class.
Code lines: 12
Contained by: module `galacticus_nodes`

function: `satelliteorbitingtidalheatingnormalizedget`
Description: Get the `tidalHeatingNormalized` property of an `orbiting` implementation of the `satellite` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `satelliteorbitingtidalheatingnormalizedjcbnzr`
Description: Indicate that the `tidalHeatingNormalized` property of an `orbiting` implementation of the `satellite` component class is inactive for differential equation solving.
Code lines: 12
Contained by: module `galacticus_nodes`

subroutine: `satelliteorbitingtidalheatingnormalizedrate`

Description: Accumulate to the rate of change of the `tidalHeatingNormalized` property of an `orbiting` implementation of the `satellite` component class.
Code lines: 31
Contained by: module galacticus_nodes

function: `satelliteorbitingtidalheatingnormalizedrateget`
Description: Get the rate of change of the `tidalHeatingNormalized` property of an `orbiting` implementation of the `satellite` component class.
Code lines: 23
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: `satelliteorbitingtidalheatingnormalizedscale`
Description: Set the absolute scale of the `tidalHeatingNormalized` property of an `orbiting` implementation of the `satellite` component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: `satelliteorbitingtidalheatingnormalizedset`
Description: Set the `tidalHeatingNormalized` property of an `orbiting` implementation of the `satellite` component class.
Code lines: 10
Contained by: module galacticus_nodes

function: `satelliteorbitingtidaltensorpathintegratedcount`
Description: Return a count of the number of scalar properties in the `tidalTensorPathIntegrated` property of an `orbiting` implementation of the `satellite` component class.
Code lines: 12
Contained by: module galacticus_nodes

function: `satelliteorbitingtidaltensorpathintegratedget`
Description: Get the `tidalTensorPathIntegrated` property of an `orbiting` implementation of the `satellite` component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: `satelliteorbitingtidaltensorpathintegratedjcbnzr`
Description: Indicate that the `tidalTensorPathIntegrated` property of an `orbiting` implementation of the `satellite` component class is inactive for differential equation solving.
Code lines: 14
Contained by: module galacticus_nodes

subroutine: `satelliteorbitingtidaltensorpathintegratedrate`
Description: Accumulate to the rate of change of the `tidalTensorPathIntegrated` property of an `orbiting` implementation of the `satellite` component class.
Code lines: 38
Contained by: module galacticus_nodes

function: `satelliteorbitingtidaltensorpathintegratedrateget`
Description: Get the rate of change of the `tidalTensorPathIntegrated` property of an `orbiting` implementation of the `satellite` component class.

Code lines: 26
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: satelliteorbitingtidaltensorpathintegratedscale
Description: Set the absolute scale of the tidalTensorPathIntegrated property of an orbiting implementation of the satellite component class.
Code lines: 15
Contained by: module galacticus_nodes

subroutine: satelliteorbitingtidaltensorpathintegratedset
Description: Set the tidalTensorPathIntegrated property of an orbiting implementation of the satellite component class.
Code lines: 10
Contained by: module galacticus_nodes

function: satelliteorbitingvelocitycount
Description: Return a count of the number of scalar properties in the velocity property of an orbiting implementation of the satellite component class.
Code lines: 13
Contained by: module galacticus_nodes

function: satelliteorbitingvelocityget
Description: Get the velocity property of an orbiting implementation of the satellite component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: satelliteorbitingvelocityjcbnzr
Description: Indicate that the velocity property of an orbiting implementation of the satellite component class is inactive for differential equation solving.
Code lines: 12
Contained by: module galacticus_nodes

subroutine: satelliteorbitingvelocityrate
Description: Accumulate to the rate of change of the velocity property of an orbiting implementation of the satellite component class.
Code lines: 33
Contained by: module galacticus_nodes

function: satelliteorbitingvelocityrateget
Description: Get the rate of change of the velocity property of an orbiting implementation of the satellite component class.
Code lines: 25
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: satelliteorbitingvelocityscale
Description: Set the absolute scale of the velocity property of an orbiting implementation of the satellite component class.
Code lines: 13

Contained by: module galacticus_nodes

subroutine: satelliteorbitingvelocityset
Description: Set the velocity property of an orbiting implementation of the satellite component class.
Code lines: 20
Contained by: module galacticus_nodes
Modules used: memory_management

function: satelliteorbitingvirialorbitget
Description: Get the virialOrbit property of an orbiting implementation of the satellite component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: satelliteorbitingvirialorbitset
Description: Set the value of the virialOrbit property of the orbiting implementation of the satellite component using a deferred function.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: satelliteorbitingvirialorbitsetfunction
Description: Set the function to be used for the set method of the virialOrbit property of the SatelliteOrbiting component.
Code lines: 10
Contained by: module galacticus_nodes

function: satelliteorbitingvirialorbitsetisattached
Description: Return true if the deferred function used to set the virialOrbit property of the SatelliteOrbiting component class has been attached.
Code lines: 8
Contained by: module galacticus_nodes

subroutine: satelliteorbitingvirialorbitsetvalue
Description: Set the virialOrbit property of an orbiting implementation of the satellite component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: satelliteoutput
Description: Populate output buffers with properties to output for a satellite component.
Code lines: 56
Contained by: module galacticus_nodes
Modules used: multi_counters

subroutine: satelliteoutputcount
Description: Increment the count of properties to output for a generic satellite component.
Code lines: 15
Contained by: module galacticus_nodes

subroutine: `satelliteoutputnames`
Description: Establish the names of properties to output for a generic `satellite` component.
Code lines: 20
Contained by: module `galacticus_nodes`

function: `satelliteposition`
Description: Returns the default value for the `position` property for the `satellite` component class.
Code lines: 18
Contained by: module `galacticus_nodes`

function: `satellitepositionattributematch`
Description: Return a text list of component implementations in the `satellite` class that have the desired attributes for the `position` property
Code lines: 28
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `satellitepositionisgettable`
Description: Returns true if the `position` property is gettable for the `satellite` component class.
Code lines: 9
Contained by: module `galacticus_nodes`

function: `satellitepositionrateget`
Description: Returns a zero rate for the `position` property for the `satellite` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

subroutine: `satellitepostoutput`
Description: Perform post-output processing of a `satellite` component.
Code lines: 14
Contained by: module `galacticus_nodes`

function: `satellitepresetboundmasshistoryget`
Description: Get the `boundMassHistory` property of an `preset` implementation of the `satellite` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `satellitepresetboundmasshistoryset`
Description: Set the `boundMassHistory` property of an `preset` implementation of the `satellite` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `satellitepresetisorphanget`
Description: Get the `isOrphan` property of an `preset` implementation of the `satellite` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `satellitepresetisorphanset`
Description: Set the `isOrphan` property of an `preset` implementation of the `satellite` component class.
Code lines: 10
Contained by: module galacticus_nodes

function: `satellitepresetmergeboundmass`
Description: Return the satellite bound mass at the current time.
Code lines: 43
Contained by: module galacticus_nodes
Modules used: `fgsl` histories `iso_c_binding` numerical_interpolation

function: `satellitepresetnodeindex`
Description: Return the satellite node index.
Code lines: 40
Contained by: module galacticus_nodes
Modules used: `fgsl` histories `iso_c_binding` kind_numbers numerical_interpolation

function: `satellitepresetnodeindexhistoryget`
Description: Get the `nodeIndexHistory` property of an `preset` implementation of the `satellite` component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: `satellitepresetnodeindexhistoryset`
Description: Set the `nodeIndexHistory` property of an `preset` implementation of the `satellite` component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: `satellitepresetoutputcount`
Description: Increment the count of properties to output for a `preset` implementation of the `satellite` component.
Code lines: 20
Contained by: module galacticus_nodes

subroutine: `satellitepresetoutputnames`
Description: Return the names of properties to output for a `preset` implementation of the `satellite` component.
Code lines: 41
Contained by: module galacticus_nodes

function: `satellitepresettimeofmergingget`
Description: Get the `timeOfMerging` property of an `preset` implementation of the `satellite` component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: `satellitepresettimeofmergingset`
Description: Set the `timeOfMerging` property of an `preset` implementation of the `satellite` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `satellitepresetvirialorbitget`
Description: Get the `virialOrbit` property of an `preset` implementation of the `satellite` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `satellitepresetvirialorbitset`
Description: Set the `virialOrbit` property of an `preset` implementation of the `satellite` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `satellitestandardboundmasscount`
Description: Return a count of the number of scalar properties in the `boundMass` property of an `standard` implementation of the `satellite` component class.
Code lines: 12
Contained by: module `galacticus_nodes`

function: `satellitestandardboundmassget`
Description: Get the `boundMass` property of an `standard` implementation of the `satellite` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `satellitestandardboundmassjcbnzr`
Description: Indicate that the `boundMass` property of an `standard` implementation of the `satellite` component class is inactive for differential equation solving.
Code lines: 12
Contained by: module `galacticus_nodes`

subroutine: `satellitestandardboundmassrate`
Description: Accumulate to the rate of change of the `boundMass` property of an `standard` implementation of the `satellite` component class.
Code lines: 31
Contained by: module `galacticus_nodes`

function: `satellitestandardboundmassrateget`
Description: Get the rate of change of the `boundMass` property of an `standard` implementation of the `satellite` component class.
Code lines: 23
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `satellitestandardboundmassscale`

Description: Set the absolute scale of the `boundMass` property of an `standard` implementation of the `satellite` component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: `satellitestandardboundmassset`
Description: Set the `boundMass` property of an `standard` implementation of the `satellite` component class.
Code lines: 10
Contained by: module galacticus_nodes

function: `satellitestandardmergetimecount`
Description: Return a count of the number of scalar properties in the `mergeTime` property of an `standard` implementation of the `satellite` component class.
Code lines: 12
Contained by: module galacticus_nodes

subroutine: `satellitestandardmergetimejcbnzr`
Description: Indicate that the `mergeTime` property of an `standard` implementation of the `satellite` component class is inactive for differential equation solving.
Code lines: 12
Contained by: module galacticus_nodes

subroutine: `satellitestandardmergetimerate`
Description: Accumulate to the rate of change of the `mergeTime` property of an `standard` implementation of the `satellite` component class.
Code lines: 31
Contained by: module galacticus_nodes

function: `satellitestandardmergetimerateget`
Description: Get the rate of change of the `mergeTime` property of an `standard` implementation of the `satellite` component class.
Code lines: 23
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: `satellitestandardmergetimescale`
Description: Set the absolute scale of the `mergeTime` property of an `standard` implementation of the `satellite` component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: `satellitestandardmergetimeset`
Description: Set the `mergeTime` property of an `standard` implementation of the `satellite` component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: `satellitestandardoutputcount`
Description: Increment the count of properties to output for a `standard` implementation of the `satellite` component.

Code lines: 20
Contained by: module galacticus_nodes

subroutine: satellitestandardoutputnames
Description: Return the names of properties to output for a standard implementation of the satellite component.
Code lines: 38
Contained by: module galacticus_nodes

subroutine: satellitestandardpostoutput
Description: Perform post-output processing for a standard implementation of the satellite component.
Code lines: 10
Contained by: module galacticus_nodes

function: satellitestandardvirialorbitget
Description: Get the value of the virialOrbit property of the standard implementation of the satellite component using a deferred function.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: satellitestandardvirialorbitgetfunction
Description: Set the function to be used for the get method of the virialOrbit property of the SatelliteStandard component.
Code lines: 10
Contained by: module galacticus_nodes

function: satellitestandardvirialorbitgetisattached
Description: Return true if the deferred function used to get the virialOrbit property of the SatelliteStandard component class has been attached.
Code lines: 8
Contained by: module galacticus_nodes

function: satellitestandardvirialorbitgetvalue
Description: Get the virialOrbit property of an standard implementation of the satellite component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: satellitestandardvirialorbitset
Description: Set the value of the virialOrbit property of the standard implementation of the satellite component using a deferred function.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: satellitestandardvirialorbitsetfunction
Description: Set the function to be used for the set method of the virialOrbit property of the SatelliteStandard component.
Code lines: 10
Contained by: module galacticus_nodes

function: `satellitestandardvirialorbitsetisattached`
Description: Return true if the deferred function used to set the `virialOrbit` property of the `SatelliteStandard` component class has been attached.
Code lines: 8
Contained by: module `galacticus_nodes`

subroutine: `satellitestandardvirialorbitsetvalue`
Description: Set the `virialOrbit` property of an `standard` implementation of the `satellite` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `satellitetidalheatingnormalized`
Description: Returns the default value for the `tidalHeatingNormalized` property for the `satellite` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `satellitetidalheatingnormalizedattributematch`
Description: Return a text list of component implementations in the `satellite` class that have the desired attributes for the `tidalHeatingNormalized` property
Code lines: 28
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `satellitetidalheatingnormalizedisgettable`
Description: Returns true if the `tidalHeatingNormalized` property is gettable for the `satellite` component class.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `satellitetidalheatingnormalizedrateget`
Description: Returns a zero rate for the `tidalHeatingNormalized` property for the `satellite` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `satellitetidaltensorpathintegrated`
Description: Returns the default value for the `tidalTensorPathIntegrated` property for the `satellite` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `satellitetidaltensorpathintegratedattributematch`
Description: Return a text list of component implementations in the `satellite` class that have the desired attributes for the `tidalTensorPathIntegrated` property
Code lines: 28
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `satellitetidaltensorpathintegratedisgettable`
Description: Returns true if the `tidalTensorPathIntegrated` property is gettable for the `satellite` component class.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `satellitetidaltensorpathintegratedrateget`
Description: Returns a zero rate for the `tidalTensorPathIntegrated` property for the `satellite` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `satellitetimeofmerging`
Description: Returns the default value for the `timeOfMerging` property for the `satellite` component class.
Code lines: 29
Contained by: module `galacticus_nodes`

function: `satellitetimeofmergingattributematch`
Description: Return a text list of component implementations in the `satellite` class that have the desired attributes for the `timeOfMerging` property
Code lines: 63
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `satellitetimeofmergingisgettable`
Description: Returns true if the `timeOfMerging` property is gettable for the `satellite` component class.
Code lines: 12
Contained by: module `galacticus_nodes`

function: `satellitetimeofmergingrateget`
Description: Returns a zero rate for the `timeOfMerging` property for the `satellite` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `satellitevelocity`
Description: Returns the default value for the `velocity` property for the `satellite` component class.
Code lines: 18
Contained by: module `galacticus_nodes`

function: `satellitevelocityattributematch`
Description: Return a text list of component implementations in the `satellite` class that have the desired attributes for the `velocity` property
Code lines: 28
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `satellitevelocityisgettable`

Description: Returns true if the `velocity` property is gettable for the `satellite` component class.
Code lines: 9
Contained by: module `galacticus_nodes`

function: `satellitevelocityrateget`
Description: Returns a zero rate for the `velocity` property for the `satellite` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `satelliteverysimplemergetimecount`
Description: Return a count of the number of scalar properties in the `mergeTime` property of an `verySimple` implementation of the `satellite` component class.
Code lines: 12
Contained by: module `galacticus_nodes`

subroutine: `satelliteverysimplemergetimejcbnzr`
Description: Indicate that the `mergeTime` property of an `verySimple` implementation of the `satellite` component class is inactive for differential equation solving.
Code lines: 12
Contained by: module `galacticus_nodes`

subroutine: `satelliteverysimplemergetimerate`
Description: Accumulate to the rate of change of the `mergeTime` property of an `verySimple` implementation of the `satellite` component class.
Code lines: 31
Contained by: module `galacticus_nodes`

function: `satelliteverysimplemergetimerateget`
Description: Get the rate of change of the `mergeTime` property of an `verySimple` implementation of the `satellite` component class.
Code lines: 23
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `satelliteverysimplemergetimescale`
Description: Set the absolute scale of the `mergeTime` property of an `verySimple` implementation of the `satellite` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

subroutine: `satelliteverysimplemergetimeset`
Description: Set the `mergeTime` property of an `verySimple` implementation of the `satellite` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `satelliteverysimpleoutputcount`
Description: Increment the count of properties to output for a `verySimple` implementation of the `satellite` component.
Code lines: 21
Contained by: module `galacticus_nodes`

subroutine: `satelliteverysimpleoutputnames`
Description: Return the names of properties to output for a `verySimple` implementation of the `satellite` component.
Code lines: 37
Contained by: module `galacticus_nodes`

function: `satellitevirialorbit`
Description: Returns the default value for the `virialOrbit` property for the `satellite` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `satellitevirialorbitattributematch`
Description: Return a text list of component implementations in the `satellite` class that have the desired attributes for the `virialOrbit` property
Code lines: 52
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `satellitevirialorbitisgettable`
Description: Returns true if the `virialOrbit` property is gettable for the `satellite` component class.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `satellitevirialorbitrateget`
Description: Returns a zero rate for the `virialOrbit` property for the `satellite` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `spheroidabundancesgas`
Description: Returns the default value for the `abundancesGas` property for the `spheroid` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `spheroidabundancesgasattributematch`
Description: Return a text list of component implementations in the `spheroid` class that have the desired attributes for the `abundancesGas` property
Code lines: 37
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `spheroidabundancesgasisgettable`
Description: Returns true if the `abundancesGas` property is gettable for the `spheroid` component class.
Code lines: 8
Contained by: module `galacticus_nodes`

subroutine: `spheroidabundancesgasrate`
Description: Accept a rate set for the `abundancesGas` property of the `spheroid` component class. Trigger an interrupt to create the component.
Code lines: 21

Contained by: module galacticus_nodes
Modules used: galacticus_error

function: spheroidabundancesgasrateget
Description: Returns a zero rate for the abundancesGas property for the spheroid component class.
Code lines: 13
Contained by: module galacticus_nodes

function: spheroidabundancesstellar
Description: Returns the default value for the abundancesStellar property for the spheroid component class.
Code lines: 13
Contained by: module galacticus_nodes

function: spheroidabundancesstellarattributematch
Description: Return a text list of component implementations in the spheroid class that have the desired attributes for the abundancesStellar property
Code lines: 37
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: spheroidabundancesstellarisgettable
Description: Returns true if the abundancesStellar property is gettable for the spheroid component class.
Code lines: 8
Contained by: module galacticus_nodes

subroutine: spheroidabundancesstellarrate
Description: Accept a rate set for the abundancesStellar property of the spheroid component class. Trigger an interrupt to create the component.
Code lines: 21
Contained by: module galacticus_nodes
Modules used: galacticus_error

function: spheroidabundancesstellarrateget
Description: Returns a zero rate for the abundancesStellar property for the spheroid component class.
Code lines: 13
Contained by: module galacticus_nodes

function: spheroidangularmomentum
Description: Returns the default value for the angularMomentum property for the spheroid component class.
Code lines: 13
Contained by: module galacticus_nodes

function: spheroidangularmomentumattributematch
Description: Return a text list of component implementations in the spheroid class that have the desired attributes for the angularMomentum property
Code lines: 28
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: `spheroidangularmomentumisgettable`
Description: Returns true if the `angularMomentum` property is gettable for the `spheroid` component class.
Code lines: 8
Contained by: module `galacticus_nodes`

subroutine: `spheroidangularmomentumrate`
Description: Accept a rate set for the `angularMomentum` property of the `spheroid` component class. Trigger an interrupt to create the component.
Code lines: 21
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

function: `spheroidangularmomentumrateget`
Description: Returns a zero rate for the `angularMomentum` property for the `spheroid` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

subroutine: `spheroidcreatebyinterrupt`
Description: Create the `spheroid` component of `self` via an interrupt.
Code lines: 14
Contained by: module `galacticus_nodes`

function: `spheroidenergygasinput`
Description: Returns the default value for the `energyGasInput` property for the `spheroid` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `spheroidenergygasinputattributematch`
Description: Return a text list of component implementations in the `spheroid` class that have the desired attributes for the `energyGasInput` property
Code lines: 30
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `spheroidenergygasinputisgettable`
Description: Returns true if the `energyGasInput` property is gettable for the `spheroid` component class.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `spheroidenergygasinputrateget`
Description: Returns a zero rate for the `energyGasInput` property for the `spheroid` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `spheroidhalfmassradius`
Description: Returns the default value for the `halfMassRadius` property for the `spheroid` component class.
Code lines: 13

Contained by: module galacticus_nodes

function: spheroidhalfmassradiusattributematch
Description: Return a text list of component implementations in the spheroid class that have the desired attributes for the halfMassRadius property
Code lines: 41
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: spheroidhalfmassradiusisgettable
Description: Returns true if the halfMassRadius property is gettable for the spheroid component class.
Code lines: 8
Contained by: module galacticus_nodes

function: spheroidhalfmassradiusrateget
Description: Returns a zero rate for the halfMassRadius property for the spheroid component class.
Code lines: 13
Contained by: module galacticus_nodes

function: spheroidisinitialized
Description: Returns the default value for the isInitialized property for the spheroid component class.
Code lines: 13
Contained by: module galacticus_nodes

function: spheroidisinitializedattributematch
Description: Return a text list of component implementations in the spheroid class that have the desired attributes for the isInitialized property
Code lines: 41
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: spheroidisinitializedisgettable
Description: Returns true if the isInitialized property is gettable for the spheroid component class.
Code lines: 8
Contained by: module galacticus_nodes

function: spheroidisinitializedrateget
Description: Returns a zero rate for the isInitialized property for the spheroid component class.
Code lines: 13
Contained by: module galacticus_nodes

function: spheroidluminositiesstellar
Description: Returns the default value for the luminositiesStellar property for the spheroid component class.
Code lines: 13
Contained by: module galacticus_nodes

function: spheroidluminositiesstellarattributematch
Description: Return a text list of component implementations in the spheroid class that have the desired attributes for the luminositiesStellar property

Code lines: 37
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: spheroidluminositiesstellarisgettable
Description: Returns true if the luminositiesStellar property is gettable for the spheroid component class.
Code lines: 8
Contained by: module galacticus_nodes

subroutine: spheroidluminositiesstellarrate
Description: Accept a rate set for the luminositiesStellar property of the spheroid component class. Trigger an interrupt to create the component.
Code lines: 21
Contained by: module galacticus_nodes
Modules used: galacticus_error

function: spheroidluminositiesstellarrateget
Description: Returns a zero rate for the luminositiesStellar property for the spheroid component class.
Code lines: 13
Contained by: module galacticus_nodes

function: spheroidmassgas
Description: Returns the default value for the massGas property for the spheroid component class.
Code lines: 13
Contained by: module galacticus_nodes

function: spheroidmassgasattributematch
Description: Return a text list of component implementations in the spheroid class that have the desired attributes for the massGas property
Code lines: 37
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: spheroidmassgasisgettable
Description: Returns true if the massGas property is gettable for the spheroid component class.
Code lines: 8
Contained by: module galacticus_nodes

subroutine: spheroidmassgasrate
Description: Accept a rate set for the massGas property of the spheroid component class. Trigger an interrupt to create the component.
Code lines: 21
Contained by: module galacticus_nodes
Modules used: galacticus_error

function: spheroidmassgasrateget
Description: Returns a zero rate for the massGas property for the spheroid component class.
Code lines: 13

Contained by: module galacticus_nodes

function: spheroidmassgassink
Description: Returns the default value for the massGasSink property for the spheroid component class.
Code lines: 13
Contained by: module galacticus_nodes

function: spheroidmassgassinkattributematch
Description: Return a text list of component implementations in the spheroid class that have the desired attributes for the massGasSink property
Code lines: 30
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: spheroidmassgassinkisgettable
Description: Returns true if the massGasSink property is gettable for the spheroid component class.
Code lines: 8
Contained by: module galacticus_nodes

function: spheroidmassgassinkrateget
Description: Returns a zero rate for the massGasSink property for the spheroid component class.
Code lines: 13
Contained by: module galacticus_nodes

function: spheroidmassstellar
Description: Returns the default value for the massStellar property for the spheroid component class.
Code lines: 13
Contained by: module galacticus_nodes

function: spheroidmassstellarattributematch
Description: Return a text list of component implementations in the spheroid class that have the desired attributes for the massStellar property
Code lines: 37
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: spheroidmassstellarformed
Description: Returns the default value for the massStellarFormed property for the spheroid component class.
Code lines: 13
Contained by: module galacticus_nodes

function: spheroidmassstellarformedattributematch
Description: Return a text list of component implementations in the spheroid class that have the desired attributes for the massStellarFormed property
Code lines: 28
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: spheroidmassstellarformedisgettable

Description: Returns true if the `massStellarFormed` property is gettable for the `spheroid` component class.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `spheroidmassstellarformedrateget`
Description: Returns a zero rate for the `massStellarFormed` property for the `spheroid` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `spheroidmassstellarisgettable`
Description: Returns true if the `massStellar` property is gettable for the `spheroid` component class.
Code lines: 8
Contained by: module `galacticus_nodes`

subroutine: `spheroidmassstellarrate`
Description: Accept a rate set for the `massStellar` property of the `spheroid` component class. Trigger an interrupt to create the component.
Code lines: 21
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

function: `spheroidmassstellarrateget`
Description: Returns a zero rate for the `massStellar` property for the `spheroid` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

subroutine: `spheroidnulloutputcount`
Description: Increment the count of properties to output for a `null` implementation of the `spheroid` component.
Code lines: 22
Contained by: module `galacticus_nodes`

subroutine: `spheroidnulloutputnames`
Description: Return the names of properties to output for a `null` implementation of the `spheroid` component.
Code lines: 39
Contained by: module `galacticus_nodes`

subroutine: `spheroidoutput`
Description: Populate output buffers with properties to output for a `spheroid` component.
Code lines: 72
Contained by: module `galacticus_nodes`
Modules used: `multi_counters`

subroutine: `spheroidoutputcount`
Description: Increment the count of properties to output for a generic `spheroid` component.
Code lines: 15
Contained by: module `galacticus_nodes`

subroutine: `spheroidoutputnames`
Description: Establish the names of properties to output for a generic `spheroid` component.
Code lines: 20
Contained by: module galacticus_nodes

subroutine: `spheroidpostoutput`
Description: Perform post-output processing of a `spheroid` component.
Code lines: 14
Contained by: module galacticus_nodes

function: `spheroidradius`
Description: Returns the default value for the `radius` property for the `spheroid` component class.
Code lines: 13
Contained by: module galacticus_nodes

function: `spheroidradiusattributematch`
Description: Return a text list of component implementations in the `spheroid` class that have the desired attributes for the `radius` property
Code lines: 41
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: `spheroidradiusisgettable`
Description: Returns true if the `radius` property is gettable for the `spheroid` component class.
Code lines: 8
Contained by: module galacticus_nodes

function: `spheroidradiusrateget`
Description: Returns a zero rate for the `radius` property for the `spheroid` component class.
Code lines: 13
Contained by: module galacticus_nodes

function: `spheroidstandardabundancesgascount`
Description: Return a count of the number of scalar properties in the `abundancesGas` property of an `standard` implementation of the `spheroid` component class.
Code lines: 12
Contained by: module galacticus_nodes

function: `spheroidstandardabundancesgasget`
Description: Get the `abundancesGas` property of an `standard` implementation of the `spheroid` component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: `spheroidstandardabundancesgasjcbnzr`
Description: Indicate that the `abundancesGas` property of an `standard` implementation of the `spheroid` component class is inactive for differential equation solving.
Code lines: 14
Contained by: module galacticus_nodes

subroutine: `spheroidstandardabundancesgasrate`
Description: Accumulate to the rate of change of the `abundancesGas` property of an `standard` implementation of the `spheroid` component class.
Code lines: 38
Contained by: module `galacticus_nodes`

function: `spheroidstandardabundancesgasrateget`
Description: Get the rate of change of the `abundancesGas` property of an `standard` implementation of the `spheroid` component class.
Code lines: 26
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `spheroidstandardabundancesgasscale`
Description: Set the absolute scale of the `abundancesGas` property of an `standard` implementation of the `spheroid` component class.
Code lines: 15
Contained by: module `galacticus_nodes`

subroutine: `spheroidstandardabundancesgasset`
Description: Set the `abundancesGas` property of an `standard` implementation of the `spheroid` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `spheroidstandardabundancesstellarcount`
Description: Return a count of the number of scalar properties in the `abundancesStellar` property of an `standard` implementation of the `spheroid` component class.
Code lines: 12
Contained by: module `galacticus_nodes`

function: `spheroidstandardabundancesstellarget`
Description: Get the `abundancesStellar` property of an `standard` implementation of the `spheroid` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `spheroidstandardabundancesstellarjcbnzr`
Description: Indicate that the `abundancesStellar` property of an `standard` implementation of the `spheroid` component class is inactive for differential equation solving.
Code lines: 14
Contained by: module `galacticus_nodes`

subroutine: `spheroidstandardabundancesstellarrate`
Description: Accumulate to the rate of change of the `abundancesStellar` property of an `standard` implementation of the `spheroid` component class.
Code lines: 38
Contained by: module `galacticus_nodes`

function: `spheroidstandardabundancesstellarrateget`
Description: Get the rate of change of the `abundancesStellar` property of an `standard` implementation of the `spheroid` component class.
Code lines: 26
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `spheroidstandardabundancesstellarscale`
Description: Set the absolute scale of the `abundancesStellar` property of an `standard` implementation of the `spheroid` component class.
Code lines: 15
Contained by: module `galacticus_nodes`

subroutine: `spheroidstandardabundancesstellarset`
Description: Set the `abundancesStellar` property of an `standard` implementation of the `spheroid` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `spheroidstandardangularmomentumcount`
Description: Return a count of the number of scalar properties in the `angularMomentum` property of an `standard` implementation of the `spheroid` component class.
Code lines: 12
Contained by: module `galacticus_nodes`

function: `spheroidstandardangularmomentumget`
Description: Get the `angularMomentum` property of an `standard` implementation of the `spheroid` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `spheroidstandardangularmomentumjcbnzr`
Description: Indicate that the `angularMomentum` property of an `standard` implementation of the `spheroid` component class is inactive for differential equation solving.
Code lines: 12
Contained by: module `galacticus_nodes`

subroutine: `spheroidstandardangularmomentumrate`
Description: Accumulate to the rate of change of the `angularMomentum` property of an `standard` implementation of the `spheroid` component class.
Code lines: 31
Contained by: module `galacticus_nodes`

function: `spheroidstandardangularmomentumrateget`
Description: Get the rate of change of the `angularMomentum` property of an `standard` implementation of the `spheroid` component class.
Code lines: 23
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `spheroidstandardangularmomentumscale`
Description: Set the absolute scale of the `angularMomentum` property of an `standard` implementation of the `spheroid` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

subroutine: `spheroidstandardangularmomentumset`
Description: Set the `angularMomentum` property of an `standard` implementation of the `spheroid` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `spheroidstandardcreatefunctionset`
Description: Set the create function for the `standard` implementation of the `spheroid` component class.
Code lines: 9
Contained by: module `galacticus_nodes`

subroutine: `spheroidstandardenergygasinputrate`
Description: Accumulate the rate of change of the `energyGasInput` property of the `standard` implementation of the `spheroid` component using a deferred function.
Code lines: 12
Contained by: module `galacticus_nodes`

subroutine: `spheroidstandardenergygasinputratefunction`
Description: Set the function to be used for the `rate` method of the `energyGasInput` property of the `SpheroidStandard` component.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `spheroidstandardenergygasinputrateisattached`
Description: Return true if the deferred function used to rate the `energyGasInput` property of the `SpheroidStandard` component class has been attached.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `spheroidstandardisinitializedget`
Description: Get the `isInitialized` property of an `standard` implementation of the `spheroid` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `spheroidstandardisinitializedset`
Description: Set the `isInitialized` property of an `standard` implementation of the `spheroid` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `spheroidstandardluminositiesstellarcount`
Description: Return a count of the number of scalar properties in the `luminositiesStellar` property of an `standard` implementation of the `spheroid` component class.

Code lines: 12
Contained by: module galacticus_nodes

function: spheroidstandardluminositiesstellarget
Description: Get the luminositiesStellar property of an standard implementation of the spheroid component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: spheroidstandardluminositiesstellarjcbnzr
Description: Indicate that the luminositiesStellar property of an standard implementation of the spheroid component class is inactive for differential equation solving.
Code lines: 14
Contained by: module galacticus_nodes

subroutine: spheroidstandardluminositiesstellarrate
Description: Accumulate to the rate of change of the luminositiesStellar property of an standard implementation of the spheroid component class.
Code lines: 38
Contained by: module galacticus_nodes

function: spheroidstandardluminositiesstellarrateget
Description: Get the rate of change of the luminositiesStellar property of an standard implementation of the spheroid component class.
Code lines: 26
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: spheroidstandardluminositiesstellarscale
Description: Set the absolute scale of the luminositiesStellar property of an standard implementation of the spheroid component class.
Code lines: 15
Contained by: module galacticus_nodes

subroutine: spheroidstandardluminositiesstellarset
Description: Set the luminositiesStellar property of an standard implementation of the spheroid component class.
Code lines: 10
Contained by: module galacticus_nodes

function: spheroidstandardmassgascount
Description: Return a count of the number of scalar properties in the massGas property of an standard implementation of the spheroid component class.
Code lines: 12
Contained by: module galacticus_nodes

function: spheroidstandardmassgasget
Description: Get the massGas property of an standard implementation of the spheroid component class.
Code lines: 10

Contained by: module galacticus_nodes

subroutine: spheroidstandardmassgasjcbnzr
Description: Indicate that the massGas property of an standard implementation of the spheroid component class is inactive for differential equation solving.
Code lines: 12
Contained by: module galacticus_nodes

subroutine: spheroidstandardmassgasrate
Description: Accumulate to the rate of change of the massGas property of an standard implementation of the spheroid component class.
Code lines: 31
Contained by: module galacticus_nodes

function: spheroidstandardmassgasrateget
Description: Get the rate of change of the massGas property of an standard implementation of the spheroid component class.
Code lines: 23
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: spheroidstandardmassgasscale
Description: Set the absolute scale of the massGas property of an standard implementation of the spheroid component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: spheroidstandardmassgasset
Description: Set the massGas property of an standard implementation of the spheroid component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: spheroidstandardmassgassinkrate
Description: Accumulate the rate of change of the massGasSink property of the standard implementation of the spheroid component using a deferred function.
Code lines: 12
Contained by: module galacticus_nodes

subroutine: spheroidstandardmassgassinkratefunction
Description: Set the function to be used for the rate method of the massGasSink property of the SpheroidStandard component.
Code lines: 10
Contained by: module galacticus_nodes

function: spheroidstandardmassgassinkrateisattached
Description: Return true if the deferred function used to rate the massGasSink property of the SpheroidStandard component class has been attached.
Code lines: 8
Contained by: module galacticus_nodes

function: `spheroidstandardmassstellarcount`
Description: Return a count of the number of scalar properties in the `massStellar` property of an `standard` implementation of the `spheroid` component class.
Code lines: 12
Contained by: module `galacticus_nodes`

function: `spheroidstandardmassstellarformedcount`
Description: Return a count of the number of scalar properties in the `massStellarFormed` property of an `standard` implementation of the `spheroid` component class.
Code lines: 12
Contained by: module `galacticus_nodes`

function: `spheroidstandardmassstellarformedget`
Description: Get the `massStellarFormed` property of an `standard` implementation of the `spheroid` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `spheroidstandardmassstellarformedjcbnzr`
Description: Indicate that the `massStellarFormed` property of an `standard` implementation of the `spheroid` component class is inactive for differential equation solving.
Code lines: 12
Contained by: module `galacticus_nodes`

subroutine: `spheroidstandardmassstellarformedrate`
Description: Accumulate to the rate of change of the `massStellarFormed` property of an `standard` implementation of the `spheroid` component class.
Code lines: 31
Contained by: module `galacticus_nodes`

function: `spheroidstandardmassstellarformedrateget`
Description: Get the rate of change of the `massStellarFormed` property of an `standard` implementation of the `spheroid` component class.
Code lines: 23
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `spheroidstandardmassstellarformedscale`
Description: Set the absolute scale of the `massStellarFormed` property of an `standard` implementation of the `spheroid` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

subroutine: `spheroidstandardmassstellarformedset`
Description: Set the `massStellarFormed` property of an `standard` implementation of the `spheroid` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `spheroidstandardmassstellarget`
Description: Get the `massStellar` property of an `standard` implementation of the `spheroid` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `spheroidstandardmassstellarjcbnzr`
Description: Indicate that the `massStellar` property of an `standard` implementation of the `spheroid` component class is inactive for differential equation solving.
Code lines: 12
Contained by: module `galacticus_nodes`

subroutine: `spheroidstandardmassstellarrate`
Description: Accumulate to the rate of change of the `massStellar` property of an `standard` implementation of the `spheroid` component class.
Code lines: 31
Contained by: module `galacticus_nodes`

function: `spheroidstandardmassstellarrateget`
Description: Get the rate of change of the `massStellar` property of an `standard` implementation of the `spheroid` component class.
Code lines: 23
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `spheroidstandardmassstellarscale`
Description: Set the absolute scale of the `massStellar` property of an `standard` implementation of the `spheroid` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

subroutine: `spheroidstandardmassstellarset`
Description: Set the `massStellar` property of an `standard` implementation of the `spheroid` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `spheroidstandardoutputcount`
Description: Increment the count of properties to output for a `standard` implementation of the `spheroid` component.
Code lines: 24
Contained by: module `galacticus_nodes`

subroutine: `spheroidstandardoutputnames`
Description: Return the names of properties to output for a `standard` implementation of the `spheroid` component.
Code lines: 71
Contained by: module `galacticus_nodes`

subroutine: `spheroidstandardpostoutput`
Description: Perform post-output processing for a `standard` implementation of the `spheroid` component.
Code lines: 12
Contained by: module galacticus_nodes

function: `spheroidstandardradiusget`
Description: Get the `radius` property of an `standard` implementation of the `spheroid` component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: `spheroidstandardradiusset`
Description: Set the `radius` property of an `standard` implementation of the `spheroid` component class.
Code lines: 10
Contained by: module galacticus_nodes

function: `spheroidstandardstarformationhistorycount`
Description: Return a count of the number of scalar properties in the `starFormationHistory` property of an `standard` implementation of the `spheroid` component class.
Code lines: 12
Contained by: module galacticus_nodes

function: `spheroidstandardstarformationhistoryget`
Description: Get the `starFormationHistory` property of an `standard` implementation of the `spheroid` component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: `spheroidstandardstarformationhistoryjcbnzr`
Description: Indicate that the `starFormationHistory` property of an `standard` implementation of the `spheroid` component class is inactive for differential equation solving.
Code lines: 14
Contained by: module galacticus_nodes

subroutine: `spheroidstandardstarformationhistoryrate`
Description: Accumulate the rate of change of the `starFormationHistory` property of the `standard` implementation of the `spheroid` component using a deferred function.
Code lines: 12
Contained by: module galacticus_nodes

subroutine: `spheroidstandardstarformationhistoryratefunction`
Description: Set the function to be used for the `rate` method of the `starFormationHistory` property of the `SpheroidStandard` component.
Code lines: 10
Contained by: module galacticus_nodes

function: `spheroidstandardstarformationhistoryrateget`
Description: Get the rate of change of the `starFormationHistory` property of an `standard` implementation of the `spheroid` component class.
Code lines: 26

Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: spheroidstandardstarformationhistoryrateintrinsic
Description: Accumulate directly (i.e. circumventing any deferred function binding) to the rate of change of the starFormationHistory property of an standard implementation of the spheroid component class.
Code lines: 38
Contained by: module galacticus_nodes

function: spheroidstandardstarformationhistoryrateisattached
Description: Return true if the deferred function used to rate the starFormationHistory property of the SpheroidStandard component class has been attached.
Code lines: 8
Contained by: module galacticus_nodes

subroutine: spheroidstandardstarformationhistoryscale
Description: Set the absolute scale of the starFormationHistory property of an standard implementation of the spheroid component class.
Code lines: 15
Contained by: module galacticus_nodes

subroutine: spheroidstandardstarformationhistoryset
Description: Set the starFormationHistory property of an standard implementation of the spheroid component class.
Code lines: 10
Contained by: module galacticus_nodes

function: spheroidstandardstarformationrateget
Description: Get the value of the starFormationRate property of the standard implementation of the spheroid component using a deferred function.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: spheroidstandardstarformationrategetfunction
Description: Set the function to be used for the get method of the starFormationRate property of the SpheroidStandard component.
Code lines: 10
Contained by: module galacticus_nodes

function: spheroidstandardstarformationrategetisattached
Description: Return true if the deferred function used to get the starFormationRate property of the SpheroidStandard component class has been attached.
Code lines: 8
Contained by: module galacticus_nodes

function: spheroidstandardstellarpropertieshistorycount
Description: Return a count of the number of scalar properties in the stellarPropertiesHistory property of an standard implementation of the spheroid component class.
Code lines: 12

Contained by: module galacticus_nodes

function: spheroidstandardstellarpropertieshistoryget
Description: Get the stellarPropertiesHistory property of an standard implementation of the spheroid component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: spheroidstandardstellarpropertieshistoryjcbnzr
Description: Indicate that the stellarPropertiesHistory property of an standard implementation of the spheroid component class is inactive for differential equation solving.
Code lines: 14
Contained by: module galacticus_nodes

subroutine: spheroidstandardstellarpropertieshistoryrate
Description: Accumulate the rate of change of the stellarPropertiesHistory property of the standard implementation of the spheroid component using a deferred function.
Code lines: 12
Contained by: module galacticus_nodes

subroutine: spheroidstandardstellarpropertieshistoryratefunction
Description: Set the function to be used for the rate method of the stellarPropertiesHistory property of the SpheroidStandard component.
Code lines: 10
Contained by: module galacticus_nodes

function: spheroidstandardstellarpropertieshistoryrateget
Description: Get the rate of change of the stellarPropertiesHistory property of an standard implementation of the spheroid component class.
Code lines: 26
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: spheroidstandardstellarpropertieshistoryrateintrinsic
Description: Accumulate directly (i.e. circumventing any deferred function binding) to the rate of change of the stellarPropertiesHistory property of an standard implementation of the spheroid component class.
Code lines: 38
Contained by: module galacticus_nodes

function: spheroidstandardstellarpropertieshistoryrateisattached
Description: Return true if the deferred function used to rate the stellarPropertiesHistory property of the SpheroidStandard component class has been attached.
Code lines: 8
Contained by: module galacticus_nodes

subroutine: spheroidstandardstellarpropertieshistoryscale
Description: Set the absolute scale of the stellarPropertiesHistory property of an standard implementation of the spheroid component class.
Code lines: 15

Contained by: module galacticus_nodes

subroutine: spheroidstandardstellarpropertieshistoryset
Description: Set the stellarPropertiesHistory property of an standard implementation of the spheroid component class.
Code lines: 10
Contained by: module galacticus_nodes

function: spheroidstandardvelocityget
Description: Get the velocity property of an standard implementation of the spheroid component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: spheroidstandardvelocityset
Description: Set the velocity property of an standard implementation of the spheroid component class.
Code lines: 10
Contained by: module galacticus_nodes

function: spheroidstarformationhistory
Description: Returns the default value for the starFormationHistory property for the spheroid component class.
Code lines: 13
Contained by: module galacticus_nodes

function: spheroidstarformationhistoryattributematch
Description: Return a text list of component implementations in the spheroid class that have the desired attributes for the starFormationHistory property
Code lines: 28
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: spheroidstarformationhistoryisgettable
Description: Returns true if the starFormationHistory property is gettable for the spheroid component class.
Code lines: 8
Contained by: module galacticus_nodes

subroutine: spheroidstarformationhistoryrate
Description: Accept a rate set for the starFormationHistory property of the spheroid component class. Trigger an interrupt to create the component.
Code lines: 21
Contained by: module galacticus_nodes
Modules used: galacticus_error

function: spheroidstarformationhistoryrateget
Description: Returns a zero rate for the starFormationHistory property for the spheroid component class.
Code lines: 13
Contained by: module galacticus_nodes

function: `spheroidstarformationrate`
Description: Returns the default value for the `starFormationRate` property for the `spheroid` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `spheroidstarformationrateattributematch`
Description: Return a text list of component implementations in the `spheroid` class that have the desired attributes for the `starFormationRate` property
Code lines: 41
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `spheroidstarformationrateisgettable`
Description: Returns true if the `starFormationRate` property is gettable for the `spheroid` component class.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `spheroidstarformationraterateget`
Description: Returns a zero rate for the `starFormationRate` property for the `spheroid` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `spheroidstellarpropertieshistory`
Description: Returns the default value for the `stellarPropertiesHistory` property for the `spheroid` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `spheroidstellarpropertieshistoryattributematch`
Description: Return a text list of component implementations in the `spheroid` class that have the desired attributes for the `stellarPropertiesHistory` property
Code lines: 37
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `spheroidstellarpropertieshistoryisgettable`
Description: Returns true if the `stellarPropertiesHistory` property is gettable for the `spheroid` component class.
Code lines: 8
Contained by: module `galacticus_nodes`

subroutine: `spheroidstellarpropertieshistoryrate`
Description: Accept a rate set for the `stellarPropertiesHistory` property of the `spheroid` component class. Trigger an interrupt to create the component.
Code lines: 21
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

function: `spheroidstellarpropertieshistoryrateget`
Description: Returns a zero rate for the `stellarPropertiesHistory` property for the `spheroid` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `spheroidvelocity`
Description: Returns the default value for the `velocity` property for the `spheroid` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `spheroidvelocityattributematch`
Description: Return a text list of component implementations in the `spheroid` class that have the desired attributes for the `velocity` property
Code lines: 41
Contained by: module `galacticus_nodes`
Modules used: `iso_varying_string`

function: `spheroidvelocityisgettable`
Description: Returns true if the `velocity` property is gettable for the `spheroid` component class.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `spheroidvelocityrateget`
Description: Returns a zero rate for the `velocity` property for the `spheroid` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

function: `spheroidverysimpleabundancesgascount`
Description: Return a count of the number of scalar properties in the `abundancesGas` property of an `verySimple` implementation of the `spheroid` component class.
Code lines: 12
Contained by: module `galacticus_nodes`

function: `spheroidverysimpleabundancesgasget`
Description: Get the `abundancesGas` property of an `verySimple` implementation of the `spheroid` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `spheroidverysimpleabundancesgasjcbnzr`
Description: Indicate that the `abundancesGas` property of an `verySimple` implementation of the `spheroid` component class is inactive for differential equation solving.
Code lines: 14
Contained by: module `galacticus_nodes`

subroutine: `spheroidverysimpleabundancesgasrate`
Description: Accumulate to the rate of change of the `abundancesGas` property of an `verySimple` implementation of the `spheroid` component class.

Code lines: 38
Contained by: module galacticus_nodes

function: spheroidverysimpleabundancesgasrateget
Description: Get the rate of change of the abundancesGas property of an verySimple implementation of the spheroid component class.
Code lines: 26
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: spheroidverysimpleabundancesgasscale
Description: Set the absolute scale of the abundancesGas property of an verySimple implementation of the spheroid component class.
Code lines: 15
Contained by: module galacticus_nodes

subroutine: spheroidverysimpleabundancesgasset
Description: Set the abundancesGas property of an verySimple implementation of the spheroid component class.
Code lines: 10
Contained by: module galacticus_nodes

function: spheroidverysimpleabundancesstellarcount
Description: Return a count of the number of scalar properties in the abundancesStellar property of an verySimple implementation of the spheroid component class.
Code lines: 12
Contained by: module galacticus_nodes

function: spheroidverysimpleabundancesstellarget
Description: Get the abundancesStellar property of an verySimple implementation of the spheroid component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: spheroidverysimpleabundancesstellarjcbnzr
Description: Indicate that the abundancesStellar property of an verySimple implementation of the spheroid component class is inactive for differential equation solving.
Code lines: 14
Contained by: module galacticus_nodes

subroutine: spheroidverysimpleabundancesstellarrate
Description: Accumulate to the rate of change of the abundancesStellar property of an verySimple implementation of the spheroid component class.
Code lines: 38
Contained by: module galacticus_nodes

function: spheroidverysimpleabundancesstellarrateget
Description: Get the rate of change of the abundancesStellar property of an verySimple implementation of the spheroid component class.
Code lines: 26

Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: spheroidverysimpleabundancesstellarscale
Description: Set the absolute scale of the abundancesStellar property of an verySimple implementation of the spheroid component class.
Code lines: 15
Contained by: module galacticus_nodes

subroutine: spheroidverysimpleabundancesstellarset
Description: Set the abundancesStellar property of an verySimple implementation of the spheroid component class.
Code lines: 10
Contained by: module galacticus_nodes

function: spheroidverysimpleisinitializedget
Description: Get the isInitialized property of an verySimple implementation of the spheroid component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: spheroidverysimpleisinitializedset
Description: Set the isInitialized property of an verySimple implementation of the spheroid component class.
Code lines: 10
Contained by: module galacticus_nodes

function: spheroidverysimpleluminositiesstellarcount
Description: Return a count of the number of scalar properties in the luminositiesStellar property of an verySimple implementation of the spheroid component class.
Code lines: 12
Contained by: module galacticus_nodes

function: spheroidverysimpleluminositiesstellarget
Description: Get the luminositiesStellar property of an verySimple implementation of the spheroid component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: spheroidverysimpleluminositiesstellarjcbnzr
Description: Indicate that the luminositiesStellar property of an verySimple implementation of the spheroid component class is inactive for differential equation solving.
Code lines: 14
Contained by: module galacticus_nodes

subroutine: spheroidverysimpleluminositiesstellarrate
Description: Accumulate to the rate of change of the luminositiesStellar property of an verySimple implementation of the spheroid component class.
Code lines: 38
Contained by: module galacticus_nodes

function: `spheroidverysimpleluminositiesstellarrateget`
Description: Get the rate of change of the `luminositiesStellar` property of an `verySimple` implementation of the `spheroid` component class.
Code lines: 26
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `spheroidverysimpleluminositiesstellarscale`
Description: Set the absolute scale of the `luminositiesStellar` property of an `verySimple` implementation of the `spheroid` component class.
Code lines: 15
Contained by: module `galacticus_nodes`

subroutine: `spheroidverysimpleluminositiesstellarset`
Description: Set the `luminositiesStellar` property of an `verySimple` implementation of the `spheroid` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `spheroidverysimplemassgascount`
Description: Return a count of the number of scalar properties in the `massGas` property of an `verySimple` implementation of the `spheroid` component class.
Code lines: 12
Contained by: module `galacticus_nodes`

function: `spheroidverysimplemassgasget`
Description: Get the `massGas` property of an `verySimple` implementation of the `spheroid` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `spheroidverysimplemassgasjcbnzr`
Description: Indicate that the `massGas` property of an `verySimple` implementation of the `spheroid` component class is inactive for differential equation solving.
Code lines: 12
Contained by: module `galacticus_nodes`

subroutine: `spheroidverysimplemassgasrate`
Description: Accumulate to the rate of change of the `massGas` property of an `verySimple` implementation of the `spheroid` component class.
Code lines: 31
Contained by: module `galacticus_nodes`

function: `spheroidverysimplemassgasrateget`
Description: Get the rate of change of the `massGas` property of an `verySimple` implementation of the `spheroid` component class.
Code lines: 23
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `spheroidverysimplemassgasscale`
Description: Set the absolute scale of the `massGas` property of an `verySimple` implementation of the `spheroid` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

subroutine: `spheroidverysimplemassgasset`
Description: Set the `massGas` property of an `verySimple` implementation of the `spheroid` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `spheroidverysimplemassstellarcount`
Description: Return a count of the number of scalar properties in the `massStellar` property of an `verySimple` implementation of the `spheroid` component class.
Code lines: 12
Contained by: module `galacticus_nodes`

function: `spheroidverysimplemassstellarget`
Description: Get the `massStellar` property of an `verySimple` implementation of the `spheroid` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `spheroidverysimplemassstellarjcbnzr`
Description: Indicate that the `massStellar` property of an `verySimple` implementation of the `spheroid` component class is inactive for differential equation solving.
Code lines: 12
Contained by: module `galacticus_nodes`

subroutine: `spheroidverysimplemassstellarrate`
Description: Accumulate to the rate of change of the `massStellar` property of an `verySimple` implementation of the `spheroid` component class.
Code lines: 31
Contained by: module `galacticus_nodes`

function: `spheroidverysimplemassstellarrateget`
Description: Get the rate of change of the `massStellar` property of an `verySimple` implementation of the `spheroid` component class.
Code lines: 23
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `spheroidverysimplemassstellarscale`
Description: Set the absolute scale of the `massStellar` property of an `verySimple` implementation of the `spheroid` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

subroutine: `spheroidverysimplemassstellarset`
Description: Set the `massStellar` property of an `verySimple` implementation of the `spheroid` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `spheroidverysimpleoutputcount`
Description: Increment the count of properties to output for a `verySimple` implementation of the `spheroid` component.
Code lines: 24
Contained by: module `galacticus_nodes`

subroutine: `spheroidverysimpleoutputnames`
Description: Return the names of properties to output for a `verySimple` implementation of the `spheroid` component.
Code lines: 65
Contained by: module `galacticus_nodes`

subroutine: `spheroidverysimplepostoutput`
Description: Perform post-output processing for a `verySimple` implementation of the `spheroid` component.
Code lines: 12
Contained by: module `galacticus_nodes`

function: `spheroidverysimpleradiusget`
Description: Get the `radius` property of an `verySimple` implementation of the `spheroid` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `spheroidverysimpleradiusset`
Description: Set the `radius` property of an `verySimple` implementation of the `spheroid` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `spheroidverysimplestarformationrateget`
Description: Get the value of the `starFormationRate` property of the `verySimple` implementation of the `spheroid` component using a deferred function.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `spheroidverysimplestarformationrategetfunction`
Description: Set the function to be used for the `get` method of the `starFormationRate` property of the `SpheroidVerySimple` component.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `spheroidverysimplestarformationrategetisattached`

Description: Return true if the deferred function used to get the `starFormationRate` property of the `SpheroidVerySimple` component class has been attached.
Code lines: 8
Contained by: module galacticus_nodes

function: `spheroidverysimplestellarpropertieshistorycount`
Description: Return a count of the number of scalar properties in the `stellarPropertiesHistory` property of an `verySimple` implementation of the `spheroid` component class.
Code lines: 12
Contained by: module galacticus_nodes

function: `spheroidverysimplestellarpropertieshistoryget`
Description: Get the `stellarPropertiesHistory` property of an `verySimple` implementation of the `spheroid` component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: `spheroidverysimplestellarpropertieshistoryjcbnzr`
Description: Indicate that the `stellarPropertiesHistory` property of an `verySimple` implementation of the `spheroid` component class is inactive for differential equation solving.
Code lines: 14
Contained by: module galacticus_nodes

subroutine: `spheroidverysimplestellarpropertieshistoryrate`
Description: Accumulate to the rate of change of the `stellarPropertiesHistory` property of an `verySimple` implementation of the `spheroid` component class.
Code lines: 38
Contained by: module galacticus_nodes

function: `spheroidverysimplestellarpropertieshistoryrateget`
Description: Get the rate of change of the `stellarPropertiesHistory` property of an `verySimple` implementation of the `spheroid` component class.
Code lines: 26
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: `spheroidverysimplestellarpropertieshistoryscale`
Description: Set the absolute scale of the `stellarPropertiesHistory` property of an `verySimple` implementation of the `spheroid` component class.
Code lines: 15
Contained by: module galacticus_nodes

subroutine: `spheroidverysimplestellarpropertieshistoryset`
Description: Set the `stellarPropertiesHistory` property of an `verySimple` implementation of the `spheroid` component class.
Code lines: 10
Contained by: module galacticus_nodes

function: `spheroidverysimplevelocityget`

Description: Get the `velocity` property of an `verySimple` implementation of the `spheroid` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `spheroidverysimplevelocityset`
Description: Set the `velocity` property of an `verySimple` implementation of the `spheroid` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `spinnulloutputcount`
Description: Increment the count of properties to output for a `null` implementation of the `spin` component.
Code lines: 22
Contained by: module `galacticus_nodes`

subroutine: `spinnulloutputnames`
Description: Return the names of properties to output for a `null` implementation of the `spin` component.
Code lines: 39
Contained by: module `galacticus_nodes`

subroutine: `spinoutput`
Description: Populate output buffers with properties to output for a `spin` component.
Code lines: 48
Contained by: module `galacticus_nodes`
Modules used: `multi_counters`

subroutine: `spinoutputcount`
Description: Increment the count of properties to output for a generic `spin` component.
Code lines: 15
Contained by: module `galacticus_nodes`

subroutine: `spinoutputnames`
Description: Establish the names of properties to output for a generic `spin` component.
Code lines: 20
Contained by: module `galacticus_nodes`

subroutine: `spinpostoutput`
Description: Perform post-output processing of a `spin` component.
Code lines: 14
Contained by: module `galacticus_nodes`

subroutine: `spinpreset3doutputcount`
Description: Increment the count of properties to output for a `preset3D` implementation of the `spin` component.
Code lines: 13
Contained by: module `galacticus_nodes`

subroutine: `spinpreset3doutputnames`

Description: Return the names of properties to output for a `preset3D` implementation of the `spin` component.
Code lines: 31
Contained by: module galacticus_nodes

subroutine: `spinpreset3dpostoutput`
Description: Perform post-output processing for a `preset3D` implementation of the `spin` component.
Code lines: 10
Contained by: module galacticus_nodes

function: `spinpreset3dspinvectorcount`
Description: Return a count of the number of scalar properties in the `spinVector` property of an `preset3D` implementation of the `spin` component class.
Code lines: 13
Contained by: module galacticus_nodes

function: `spinpreset3dspinvectorget`
Description: Get the `spinVector` property of an `preset3D` implementation of the `spin` component class.
Code lines: 10
Contained by: module galacticus_nodes

function: `spinpreset3dspinvectorgrowthrateget`
Description: Get the `spinVectorGrowthRate` property of an `preset3D` implementation of the `spin` component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: `spinpreset3dspinvectorgrowthrateset`
Description: Set the `spinVectorGrowthRate` property of an `preset3D` implementation of the `spin` component class.
Code lines: 20
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: `spinpreset3dspinvectorjcbnzr`
Description: Indicate that the `spinVector` property of an `preset3D` implementation of the `spin` component class is inactive for differential equation solving.
Code lines: 12
Contained by: module galacticus_nodes

subroutine: `spinpreset3dspinvectorrate`
Description: Accumulate to the rate of change of the `spinVector` property of an `preset3D` implementation of the `spin` component class.
Code lines: 33
Contained by: module galacticus_nodes

function: `spinpreset3dspinvectorrateget`
Description: Get the rate of change of the `spinVector` property of an `preset3D` implementation of the `spin` component class.
Code lines: 25

Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: spinpreset3dspinvectorscale
Description: Set the absolute scale of the spinVector property of an preset3D implementation of the spin component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: spinpreset3dspinvectorset
Description: Set the spinVector property of an preset3D implementation of the spin component class.
Code lines: 20
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: spinpresetoutputcount
Description: Increment the count of properties to output for a preset implementation of the spin component.
Code lines: 21
Contained by: module galacticus_nodes

subroutine: spinpresetoutputnames
Description: Return the names of properties to output for a preset implementation of the spin component.
Code lines: 37
Contained by: module galacticus_nodes

function: spinpresetspincount
Description: Return a count of the number of scalar properties in the spin property of an preset implementation of the spin component class.
Code lines: 12
Contained by: module galacticus_nodes

function: spinpresetspinget
Description: Get the spin property of an preset implementation of the spin component class.
Code lines: 10
Contained by: module galacticus_nodes

function: spinpresetspingrowthrateget
Description: Get the spinGrowthRate property of an preset implementation of the spin component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: spinpresetspingrowthrateset
Description: Set the spinGrowthRate property of an preset implementation of the spin component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: `spinpresetspinjcbnzr`
Description: Indicate that the `spin` property of an `preset` implementation of the `spin` component class is inactive for differential equation solving.
Code lines: 12
Contained by: module `galacticus_nodes`

subroutine: `spinpresetspinrate`
Description: Accumulate to the rate of change of the `spin` property of an `preset` implementation of the `spin` component class.
Code lines: 31
Contained by: module `galacticus_nodes`

function: `spinpresetspinrateget`
Description: Get the rate of change of the `spin` property of an `preset` implementation of the `spin` component class.
Code lines: 23
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `spinpresetspinscale`
Description: Set the absolute scale of the `spin` property of an `preset` implementation of the `spin` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

subroutine: `spinpresetspinset`
Description: Set the `spin` property of an `preset` implementation of the `spin` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `spinrandomoutputcount`
Description: Increment the count of properties to output for a `random` implementation of the `spin` component.
Code lines: 21
Contained by: module `galacticus_nodes`

subroutine: `spinrandomoutputnames`
Description: Return the names of properties to output for a `random` implementation of the `spin` component.
Code lines: 37
Contained by: module `galacticus_nodes`

function: `spinrandomspincount`
Description: Return a count of the number of scalar properties in the `spin` property of an `random` implementation of the `spin` component class.
Code lines: 12
Contained by: module `galacticus_nodes`

function: `spinrandomspinget`

Description: Get the `spin` property of an `random` implementation of the `spin` component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: `spinrandomspinjcbnzr`
Description: Indicate that the `spin` property of an `random` implementation of the `spin` component class is inactive for differential equation solving.
Code lines: 12
Contained by: module galacticus_nodes

subroutine: `spinrandomspinrate`
Description: Accumulate to the rate of change of the `spin` property of an `random` implementation of the `spin` component class.
Code lines: 31
Contained by: module galacticus_nodes

function: `spinrandomspinrateget`
Description: Get the rate of change of the `spin` property of an `random` implementation of the `spin` component class.
Code lines: 23
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: `spinrandomspinscale`
Description: Set the absolute scale of the `spin` property of an `random` implementation of the `spin` component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: `spinrandomspinset`
Description: Set the `spin` property of an `random` implementation of the `spin` component class.
Code lines: 10
Contained by: module galacticus_nodes

function: `spinspin`
Description: Returns the default value for the `spin` property for the `spin` component class.
Code lines: 13
Contained by: module galacticus_nodes

function: `spinspinattributematch`
Description: Return a text list of component implementations in the `spin` class that have the desired attributes for the `spin` property
Code lines: 55
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: `spinspingrowthrate`
Description: Returns the default value for the `spinGrowthRate` property for the `spin` component class.
Code lines: 13

Contained by: module galacticus_nodes

function: spinspingrowthrateattributematch
Description: Return a text list of component implementations in the spin class that have the desired attributes for the spinGrowthRate property
Code lines: 63
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: spinspingrowthrateisgettable
Description: Returns true if the spinGrowthRate property is gettable for the spin component class.
Code lines: 8
Contained by: module galacticus_nodes

function: spinspingrowthraterateget
Description: Returns a zero rate for the spinGrowthRate property for the spin component class.
Code lines: 13
Contained by: module galacticus_nodes

function: spinspinisgettable
Description: Returns true if the spin property is gettable for the spin component class.
Code lines: 8
Contained by: module galacticus_nodes

function: spinspinrateget
Description: Returns a zero rate for the spin property for the spin component class.
Code lines: 13
Contained by: module galacticus_nodes

function: spinspinvector
Description: Returns the default value for the spinVector property for the spin component class.
Code lines: 23
Contained by: module galacticus_nodes

function: spinspinvectorattributematch
Description: Return a text list of component implementations in the spin class that have the desired attributes for the spinVector property
Code lines: 37
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: spinspinvectorgrowthrate
Description: Returns the default value for the spinVectorGrowthRate property for the spin component class.
Code lines: 13
Contained by: module galacticus_nodes

function: spinspinvectorgrowthrateattributematch
Description: Return a text list of component implementations in the spin class that have the desired attributes for the spinVectorGrowthRate property

Code lines: 30
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: spinspinvectorgrowthrateisgettable
Description: Returns true if the spinVectorGrowthRate property is gettable for the spin component class.
Code lines: 8
Contained by: module galacticus_nodes

function: spinspinvectorgrowthraterateget
Description: Returns a zero rate for the spinVectorGrowthRate property for the spin component class.
Code lines: 13
Contained by: module galacticus_nodes

function: spinspinvectorisgettable
Description: Returns true if the spinVector property is gettable for the spin component class.
Code lines: 10
Contained by: module galacticus_nodes

function: spinspinvectorrateget
Description: Returns a zero rate for the spinVector property for the spin component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: spinvitvitskaoutputcount
Description: Increment the count of properties to output for a vitvitska implementation of the spin component.
Code lines: 21
Contained by: module galacticus_nodes

subroutine: spinvitvitskaoutputnames
Description: Return the names of properties to output for a vitvitska implementation of the spin component.
Code lines: 51
Contained by: module galacticus_nodes

function: spinvitvitskaspincount
Description: Return a count of the number of scalar properties in the spin property of an vitvitska implementation of the spin component class.
Code lines: 12
Contained by: module galacticus_nodes

function: spinvitvitskaspinget
Description: Get the value of the spin property of the vitvitska implementation of the spin component using a deferred function.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: spinvitvitskaspingetfunction

Description: Set the function to be used for the `get` method of the `spin` property of the `SpinVitvitska` component.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `spinvitvitskaspingetisattached`
Description: Return true if the deferred function used to get the `spin` property of the `SpinVitvitska` component class has been attached.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `spinvitvitskaspingetvalue`
Description: Get the `spin` property of an `vitvitska` implementation of the `spin` component class.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `spinvitvitskaspingrowthrateget`
Description: Get the value of the `spinGrowthRate` property of the `vitvitska` implementation of the `spin` component using a deferred function.
Code lines: 10
Contained by: module `galacticus_nodes`

subroutine: `spinvitvitskaspingrowthrategetfunction`
Description: Set the function to be used for the `get` method of the `spinGrowthRate` property of the `SpinVitvitska` component.
Code lines: 10
Contained by: module `galacticus_nodes`

function: `spinvitvitskaspingrowthrategetisattached`
Description: Return true if the deferred function used to get the `spinGrowthRate` property of the `SpinVitvitska` component class has been attached.
Code lines: 8
Contained by: module `galacticus_nodes`

subroutine: `spinvitvitskaspinjcbnzr`
Description: Indicate that the `spin` property of an `vitvitska` implementation of the `spin` component class is inactive for differential equation solving.
Code lines: 12
Contained by: module `galacticus_nodes`

subroutine: `spinvitvitskaspinrate`
Description: Accumulate to the rate of change of the `spin` property of an `vitvitska` implementation of the `spin` component class.
Code lines: 31
Contained by: module `galacticus_nodes`

function: `spinvitvitskaspinrateget`
Description: Get the rate of change of the `spin` property of an `vitvitska` implementation of the `spin` component class.
Code lines: 23

Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: spinvitvitskaspinscale
Description: Set the absolute scale of the spin property of an vitvitska implementation of the spin component class.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: spinvitvitskaspinset
Description: Set the spin property of an vitvitska implementation of the spin component class.
Code lines: 10
Contained by: module galacticus_nodes

function: spinvitvitskaspinvectorcount
Description: Return a count of the number of scalar properties in the spinVector property of an vitvitska implementation of the spin component class.
Code lines: 13
Contained by: module galacticus_nodes

function: spinvitvitskaspinvectorget
Description: Get the value of the spinVector property of the vitvitska implementation of the spin component using a deferred function.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: spinvitvitskaspinvectorgetfunction
Description: Set the function to be used for the get method of the spinVector property of the SpinVitvitska component.
Code lines: 10
Contained by: module galacticus_nodes

function: spinvitvitskaspinvectorgetisattached
Description: Return true if the deferred function used to get the spinVector property of the SpinVitvitska component class has been attached.
Code lines: 8
Contained by: module galacticus_nodes

function: spinvitvitskaspinvectorgetvalue
Description: Get the spinVector property of an vitvitska implementation of the spin component class.
Code lines: 10
Contained by: module galacticus_nodes

subroutine: spinvitvitskaspinvectorjcbnzr
Description: Indicate that the spinVector property of an vitvitska implementation of the spin component class is inactive for differential equation solving.
Code lines: 12
Contained by: module galacticus_nodes

subroutine: spinvitvitskaspinvectorrate

Description: Accumulate to the rate of change of the `spinVector` property of an `vitvitska` implementation of the `spin` component class.
Code lines: 33
Contained by: module `galacticus_nodes`

function: `spinvitvitskaspinvectorrateget`
Description: Get the rate of change of the `spinVector` property of an `vitvitska` implementation of the `spin` component class.
Code lines: 25
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `spinvitvitskaspinvectorscale`
Description: Set the absolute scale of the `spinVector` property of an `vitvitska` implementation of the `spin` component class.
Code lines: 13
Contained by: module `galacticus_nodes`

subroutine: `spinvitvitskaspinvectorset`
Description: Set the `spinVector` property of an `vitvitska` implementation of the `spin` component class.
Code lines: 20
Contained by: module `galacticus_nodes`
Modules used: `memory_management`

subroutine: `tree_node_attach_event`
Description: Create a new event in a tree node.
Code lines: 21
Contained by: module `galacticus_nodes`

function: `tree_node_constructor`
Description: Return a pointer to a newly created and initialized `treeNode`.
Code lines: 16
Contained by: module `galacticus_nodes`

function: `tree_node_get_earliest_progenitor`
Description: Returns a pointer to the earliest progenitor of `self`.
Code lines: 18
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

function: `tree_node_get_last_satellite`
Description: Returns a pointer to the final satellite node associated with `self`.
Code lines: 11
Contained by: module `galacticus_nodes`

function: `tree_node_index`
Description: Returns the index of a `treeNode`.
Code lines: 21
Contained by: module `galacticus_nodes`

Modules used: galacticus_error

subroutine: tree_node_index_set
Description: Sets the index of a treeNode.
Code lines: 8
Contained by: module galacticus_nodes

function: tree_node_is_on_main_branch
Description: Returns true if self is on the main branch.
Code lines: 21
Contained by: module galacticus_nodes
Modules used: galacticus_error

function: tree_node_is_primary_progenitor
Description: Returns true if self is the primary progenitor of its parent node.
Code lines: 18
Contained by: module galacticus_nodes
Modules used: galacticus_error

function: tree_node_is_primary_progenitor_of_index
Description: Return true if self is a progenitor of the node with index targetNodeIndex.
Code lines: 25
Contained by: module galacticus_nodes
Modules used: galacticus_error

function: tree_node_is_primary_progenitor_of_node
Description: Return true if self is a progenitor of targetNode.
Code lines: 25
Contained by: module galacticus_nodes
Modules used: galacticus_error

function: tree_node_is_progenitor_of_index
Description: Return true if self is a progenitor of the node with index targetNodeIndex.
Code lines: 24
Contained by: module galacticus_nodes
Modules used: galacticus_error

function: tree_node_is_progenitor_of_node
Description: Return true if self is a progenitor of targetNode.
Code lines: 24
Contained by: module galacticus_nodes
Modules used: galacticus_error

function: tree_node_is_satellite
Description: Returns true if self is a satellite.
Code lines: 31
Contained by: module galacticus_nodes
Modules used: galacticus_error

function: `tree_node_merges_with_node`
Description: Returns a pointer to the node with which `thisNode` will merge.
Code lines: 15
Contained by: module `galacticus_nodes`

subroutine: `tree_node_remove_from_host`
Description: Remove `self` from the linked list of its host node's satellites.
Code lines: 42
Contained by: module `galacticus_nodes`
Modules used: `galacticus_display` `galacticus_error` `string_handling`

subroutine: `tree_node_remove_from_mergee`
Description: Remove `self` from the linked list of its host node's satellites.
Code lines: 42
Contained by: module `galacticus_nodes`
Modules used: `galacticus_display` `galacticus_error` `string_handling`

subroutine: `tree_node_remove_paired_event`
Description: Removed a paired event from `self`. Matching is done on the basis of event ID.
Code lines: 33
Contained by: module `galacticus_nodes`

function: `tree_node_time_step`
Description: Returns the time-step last used by a `treeNode`.
Code lines: 7
Contained by: module `galacticus_nodes`

subroutine: `tree_node_time_step_set`
Description: Sets the time-step used by a `treeNode`.
Code lines: 8
Contained by: module `galacticus_nodes`

function: `tree_node_type`
Description: Returns the name of a `treeNode` object.
Code lines: 9
Contained by: module `galacticus_nodes`

function: `tree_node_unique_id`
Description: Returns the unique ID of a `treeNode`.
Code lines: 21
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

subroutine: `tree_node_unique_id_set`
Description: Sets the index of a `treeNode`.
Code lines: 16

Contained by: module galacticus_nodes

type: `treeevent`
Description: Type for events attached to trees.
Code lines: 11
Contained by: module galacticus_nodes

type: `treenode`
Description: A class for nodes in merger trees.
Code lines: 1147
Contained by: module galacticus_nodes

function: `treenodeagestatisticscount`
Description: Returns the number of `ageStatistics` components in the node.
Code lines: 26
Contained by: module galacticus_nodes
Modules used: galacticus_error

function: `treenodeagestatisticsget`
Description: Return a `ageStatistics` component member of the node. If no `instance` is specified, return the first instance. If `autoCreate` is `true` then create a single instance of the component if none exists in the node.
Code lines: 29
Contained by: module galacticus_nodes

function: `treenodebasiccount`
Description: Returns the number of `basic` components in the node.
Code lines: 26
Contained by: module galacticus_nodes
Modules used: galacticus_error

function: `treenodebasicget`
Description: Return a `basic` component member of the node. If no `instance` is specified, return the first instance. If `autoCreate` is `true` then create a single instance of the component if none exists in the node.
Code lines: 29
Contained by: module galacticus_nodes

function: `treenodeblackholecount`
Description: Returns the number of `blackHole` components in the node.
Code lines: 26
Contained by: module galacticus_nodes
Modules used: galacticus_error

function: `treenodeblackholeget`
Description: Return a `blackHole` component member of the node. If no `instance` is specified, return the first instance. If `autoCreate` is `true` then create a single instance of the component if none exists in the node.
Code lines: 29
Contained by: module galacticus_nodes

subroutine: `treenodecomponentbuilder`

Description: Build components in a `treeNode` object given an XML definition.
Code lines: 423
Contained by: module galacticus_nodes
Modules used: `fox_dom` hashes

subroutine: `treenodecomponentsmove`
Description: Move components from `self` to `targetNode`.
Code lines: 226
Contained by: module galacticus_nodes

subroutine: `treenodecopynodeto`
Description: Make a copy of `self` in `targetNode`.
Code lines: 596
Contained by: module galacticus_nodes

function: `treenodedarkmatterprofilecount`
Description: Returns the number of `darkMatterProfile` components in the node.
Code lines: 26
Contained by: module galacticus_nodes
Modules used: galacticus_error

function: `treenodedarkmatterprofileget`
Description: Return a `darkMatterProfile` component member of the node. If no `instance` is specified, return the first instance. If `autoCreate` is `true` then create a single instance of the component if none exists in the node.
Code lines: 29
Contained by: module galacticus_nodes

subroutine: `treenodedeserializeratestoarray`
Description: Deserialize rates of evolvable properties of a node from an array.
Code lines: 21
Contained by: module galacticus_nodes

subroutine: `treenodedeserializeraw`
Description: Deserialize a tree node object from a raw (binary) file.
Code lines: 426
Contained by: module galacticus_nodes

subroutine: `treenodedeserializevaluesfromarray`
Description: Deserialize evolvable properties of a node from an array.
Code lines: 212
Contained by: module galacticus_nodes
Modules used: memory_management

subroutine: `treenodedestroy`
Description: Destroy a `treeNode` object.
Code lines: 64
Contained by: module galacticus_nodes

subroutine: `treenodedestroybranch`
Description: Destroy the tree branch rooted at this given node.
Code lines: 29
Contained by: module galacticus_nodes

function: `treenodediskcount`
Description: Returns the number of `disk` components in the node.
Code lines: 26
Contained by: module galacticus_nodes
Modules used: galacticus_error

function: `treenodediskget`
Description: Return a `disk` component member of the node. If no `instance` is specified, return the first instance. If `autoCreate` is `true` then create a single instance of the component if none exists in the node.
Code lines: 29
Contained by: module galacticus_nodes

function: `treenodedynamicsstatisticscount`
Description: Returns the number of `dynamicsStatistics` components in the node.
Code lines: 26
Contained by: module galacticus_nodes
Modules used: galacticus_error

function: `treenodedynamicsstatisticsget`
Description: Return a `dynamicsStatistics` component member of the node. If no `instance` is specified, return the first instance. If `autoCreate` is `true` then create a single instance of the component if none exists in the node.
Code lines: 29
Contained by: module galacticus_nodes

function: `treenodeformationtimecount`
Description: Returns the number of `formationTime` components in the node.
Code lines: 26
Contained by: module galacticus_nodes
Modules used: galacticus_error

function: `treenodeformationtimeget`
Description: Return a `formationTime` component member of the node. If no `instance` is specified, return the first instance. If `autoCreate` is `true` then create a single instance of the component if none exists in the node.
Code lines: 29
Contained by: module galacticus_nodes

function: `treenodehosthistorycount`
Description: Returns the number of `hostHistory` components in the node.
Code lines: 26
Contained by: module galacticus_nodes

Modules used: galacticus_error

function: `treenodehosthistoryget`
Description: Return a `hostHistory` component member of the node. If no `instance` is specified, return the first instance. If `autoCreate` is `true` then create a single instance of the component if none exists in the node.
Code lines: 29
Contained by: module galacticus_nodes

function: `treenodehothalocount`
Description: Returns the number of `hotHalo` components in the node.
Code lines: 26
Contained by: module galacticus_nodes
Modules used: galacticus_error

function: `treenodehothaloget`
Description: Return a `hotHalo` component member of the node. If no `instance` is specified, return the first instance. If `autoCreate` is `true` then create a single instance of the component if none exists in the node.
Code lines: 29
Contained by: module galacticus_nodes

function: `treenodeindicescount`
Description: Returns the number of `indices` components in the node.
Code lines: 26
Contained by: module galacticus_nodes
Modules used: galacticus_error

function: `treenodeindicesget`
Description: Return a `indices` component member of the node. If no `instance` is specified, return the first instance. If `autoCreate` is `true` then create a single instance of the component if none exists in the node.
Code lines: 29
Contained by: module galacticus_nodes

subroutine: `treenodeinitialize`
Description: Initialize a `treeNode` object.
Code lines: 80
Contained by: module galacticus_nodes
Modules used: galacticus_error

function: `treenodeinteroutputcount`
Description: Returns the number of `interOutput` components in the node.
Code lines: 26
Contained by: module galacticus_nodes
Modules used: galacticus_error

function: `treenodeinteroutputget`
Description: Return a `interOutput` component member of the node. If no `instance` is specified, return the first instance. If `autoCreate` is `true` then create a single instance of the component if none exists in the node.
Code lines: 29

Contained by: module galacticus_nodes

type: `treenodelinkedlist`
Description: Type to give a linked list of treeNodes.
Code lines: 4
Contained by: module galacticus_nodes

type: `treenodelist`
Description: Type to give a list of treeNodes.
Code lines: 3
Contained by: module galacticus_nodes

function: `treenodemapdouble0`
Description: Map a rank-0, double function over components of the node.
Code lines: 242
Contained by: module galacticus_nodes
Modules used: galacticus_error

subroutine: `treenodemapvoid`
Description: Map a void function over components of the node.
Code lines: 64
Contained by: module galacticus_nodes

function: `treenodemassflowstatisticscount`
Description: Returns the number of `massFlowStatistics` components in the node.
Code lines: 26
Contained by: module galacticus_nodes
Modules used: galacticus_error

function: `treenodemassflowstatisticsget`
Description: Return a `massFlowStatistics` component member of the node. If no `instance` is specified, return the first instance. If `autoCreate` is `true` then create a single instance of the component if none exists in the node.
Code lines: 29
Contained by: module galacticus_nodes

function: `treenodemergingstatisticscount`
Description: Returns the number of `mergingStatistics` components in the node.
Code lines: 26
Contained by: module galacticus_nodes
Modules used: galacticus_error

function: `treenodemergingstatisticsget`
Description: Return a `mergingStatistics` component member of the node. If no `instance` is specified, return the first instance. If `autoCreate` is `true` then create a single instance of the component if none exists in the node.
Code lines: 29
Contained by: module galacticus_nodes

function: `treenodenbodycount`

Description: Returns the number of `nBody` components in the node.
Code lines: 26
Contained by: module galacticus_nodes
Modules used: galacticus_error

function: `treenodenbodyget`
Description: Return a `nBody` component member of the node. If no `instance` is specified, return the first instance. If `autoCreate` is `true` then create a single instance of the component if none exists in the node.
Code lines: 29
Contained by: module galacticus_nodes

subroutine: `treenodeodestepinactivesinitialize`
Description: Initialize the inactives in components of tree node `self` in preparation for an ODE solver step.
Code lines: 12
Contained by: module galacticus_nodes

subroutine: `treenodeodestepratesinitialize`
Description: Initialize the rates in components of tree node `self` in preparation for an ODE solver step.
Code lines: 12
Contained by: module galacticus_nodes

subroutine: `treenodeodestepscalesinitialize`
Description: Initialize the scales in components of tree node `self` in preparation for an ODE solver step.
Code lines: 12
Contained by: module galacticus_nodes

subroutine: `treenodeoutput`
Description: Populate output buffers with properties to output for a `treeNode`.
Code lines: 107
Contained by: module galacticus_nodes
Modules used: multi_counters

subroutine: `treenodeoutputcount`
Description: Increment the count of properties to output for a `treeNode`.
Code lines: 101
Contained by: module galacticus_nodes

subroutine: `treenodeoutputnames`
Description: Establish the names of properties to output for a `treeNode`.
Code lines: 106
Contained by: module galacticus_nodes

function: `treenodepositioncount`
Description: Returns the number of `position` components in the node.
Code lines: 26
Contained by: module galacticus_nodes
Modules used: galacticus_error

function: `treenodepositionget`
Description: Return a `position` component member of the node. If no `instance` is specified, return the first instance. If `autoCreate` is `true` then create a single instance of the component if none exists in the node.
Code lines: 29
Contained by: module galacticus_nodes

subroutine: `treenodepostoutput`
Description: Perform post-output processing of a `treeNode`.
Code lines: 100
Contained by: module galacticus_nodes

function: `treenodepropertynamefromindex`
Description: Return the name of a property given its index within an array.
Code lines: 124
Contained by: module galacticus_nodes
Modules used: iso_varying_string

function: `treenodesatellitecount`
Description: Returns the number of `satellite` components in the node.
Code lines: 26
Contained by: module galacticus_nodes
Modules used: galacticus_error

function: `treenodesatelliteget`
Description: Return a `satellite` component member of the node. If no `instance` is specified, return the first instance. If `autoCreate` is `true` then create a single instance of the component if none exists in the node.
Code lines: 29
Contained by: module galacticus_nodes

subroutine: `treenodeserializeascii`
Description: Serialize node content to ASCII.
Code lines: 137
Contained by: module galacticus_nodes
Modules used: galacticus_display iso_varying_string string_handling

function: `treenodeserializecount`
Description: Return a count of the size of the node when serialized to an array.
Code lines: 101
Contained by: module galacticus_nodes

subroutine: `treenodeserializeinactivetoarray`
Description: Serialize inactive statuses of evolvable properties of a node into an array.
Code lines: 13
Contained by: module galacticus_nodes

subroutine: `treenodeserializeoffsets`
Description: Compute offsets into serialization arrays for `treeNode` object.
Code lines: 124
Contained by: module `galacticus_nodes`

subroutine: `treenodeserializeratestoarray`
Description: Serialize rates of evolvable properties of a node into an array.
Code lines: 21
Contained by: module `galacticus_nodes`

subroutine: `treenodeserializeraw`
Description: Serialize all content of a tree node to a raw (binary) file.
Code lines: 245
Contained by: module `galacticus_nodes`

subroutine: `treenodeserializescaletoarray`
Description: Serialize scales of evolvable properties of a node into an array.
Code lines: 21
Contained by: module `galacticus_nodes`

subroutine: `treenodeserializevaluestoarray`
Description: Serialize evolvable properties of a node into an array.
Code lines: 140
Contained by: module `galacticus_nodes`
Modules used: `memory_management`

subroutine: `treenodeserializexml`
Description: Serialize tree node content as XML.
Code lines: 129
Contained by: module `galacticus_nodes`
Modules used: `galacticus_display` `iso_varying_string` `string_handling`

function: `treenodesizeof`
Description: Compute the size (in bytes) of the tree node.
Code lines: 106
Contained by: module `galacticus_nodes`

function: `treenodespheroidcount`
Description: Returns the number of `spheroid` components in the node.
Code lines: 26
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

function: `treenodespheroidget`
Description: Return a `spheroid` component member of the node. If no `instance` is specified, return the first instance. If `autoCreate` is `true` then create a single instance of the component if none exists in the node.
Code lines: 29
Contained by: module `galacticus_nodes`

function: `treenodespincount`
Description: Returns the number of `spin` components in the node.
Code lines: 26
Contained by: module `galacticus_nodes`
Modules used: `galacticus_error`

function: `treenodespinget`
Description: Return a `spin` component member of the node. If no `instance` is specified, return the first instance. If `autoCreate` is `true` then create a single instance of the component if none exists in the node.
Code lines: 29
Contained by: module `galacticus_nodes`

function: `treenodewalkbranchwithsatellites`
Description: This function provides a mechanism for walking through the branches of the merger tree. Given a pointer `self` to a branch of the tree, it will return the next node that should be visited in the tree. Thus, if `self` is initially set to the base of the merger tree and `Merger_Tree_Walk_Branch()` is called repeatedly it will walk through every node of the branch. Once the entire branch has been walked, a `null()` pointer will be returned, indicating that there are no more nodes to walk. Each node will be visited once and once only if the branch is walked in this way. Note that it is important that the walk descends to satellites before descending to children: the routines that destroy merger tree branches rely on this since child nodes are used in testing whether a node is a satellite—if they are destroyed prior to the test being made then problems with dangling pointers will occur.
Code lines: 55
Contained by: module `galacticus_nodes`

function: `treenodewalktreewithsatellites`
Description: Merger tree walk function which also descends through satellite nodes. Note that it is important that the walk descends to satellites before descending to children: the routines that destroy merger tree branches rely on this since child nodes are used in testing whether a node is a satellite—if they are destroyed prior to the test being made then problems with dangling pointers will occur.
Code lines: 46
Contained by: module `galacticus_nodes`

type: `universe`
Description: The universe object class.
Code lines: 43
Contained by: module `galacticus_nodes`

function: `universecreateevent`
Description: Create a new event in a universe.
Code lines: 24
Contained by: module `galacticus_nodes`

type: `universeevent`
Description: Type for events attached to universes.
Code lines: 12

Contained by: module galacticus_nodes

function: universepoptree
Description: Pop a tree from the universe.
Code lines: 19
Contained by: module galacticus_nodes

subroutine: universepushtree
Description: Pop a tree from the universe.
Code lines: 23
Contained by: module galacticus_nodes

subroutine: universeremoveevent
Description: Remove an event from self.
Code lines: 31
Contained by: module galacticus_nodes

file: objects.nodes.components.F90
Description: Contains a module which implements top-level functions for node components.
Code lines: 309

module: node_components
Description: Implements top-level functions for node components.
Code lines: 287
Contained by: file objects.nodes.components.F90
Used by:

subroutine particulateoperate	function catalogprojectedcorrelationfunctionconstructorpa
subroutine catalogprojectedcorrelationfunctiondestructor	function evolveforestsconstructorparameters
subroutine evolveforestsdestructor	subroutine evolveforestsperform
function excursionsetsconstructorparameters	subroutine excursionsetsdestructor
function halomassfunctionconstructorparameters	subroutine halomassfunctiondestructor
function halomodelprojectedcorrelationfunctionconstructorparameters	subroutine halomodelprojectedcorrelationfunctiondestructor
function halomodelgenerateconstructorparameters	subroutine halomodelgeneratedestructor
function halospindistributionconstructorparameters	subroutine halospindistributiondestructor
function posteriorsampleconstructorparameters	subroutine posteriorsampledestructor
program test_diemerkravtsov2014_concentration	program test_nfw96_concentration_dark_energy
program test_prada2011_concentration	program test_zhao2009_flat
program test_zhao2009_dark_energy	program test_zhao2009_open
program test_correa2015_concentration	program test_dark_matter_halo_radius_enclosing_mass

program test_dark_matter_profiles

program test_dark_matter_profiles_generic

program test_correa2015_mah

subroutine: node_components_initialize

Description: Perform initialization tasks for node components.

Code lines: 90

Contained by: module node_components

Modules used: input_parameters, node_component_age_statistics_standard, node_component_basic_standard_extended, node_component_black_hole_noncentral, node_component_black_hole_simple, node_component_black_hole_standard, node_component_dark_matter_profile_scale, node_component_dark_matter_profile_scale_shape, node_component_disk_standard, node_component_disk_very_simple, node_component_disk_very_simple_size, node_component_dynamics_statistics_bars, node_component_formation_times_cole2000, node_component_formation_times_mass_fraction, node_component_hot_halo_cold_mode, node_component_hot_halo_standard, node_component_hot_halo_vs_delayed, node_component_mass_flow_statistics_standard, node_component_merging_statistics_recent, node_component_merging_statistics_standard, node_component_nbody_generic, node_component_position_preset, node_component_position_preset_orphans, node_component_satellite_standard, node_component_satellite_very_simple, node_component_spheroid_standard, node_component_spheroid_very_simple, node_component_spin_random, node_component_spin_vitvitska

subroutine: node_components_thread_initialize

Description: Perform per-thread initialization tasks for node components.

Code lines: 88

Contained by: module node_components

Modules used: input_parameters, node_component_basic_standard_extended, node_component_black_hole_noncentral, node_component_black_hole_simple, node_component_black_hole_standard, node_component_dark_matter_profile_scale, node_component_dark_matter_profile_scale_shape, node_component_disk_standard, node_component_disk_very_simple, node_component_disk_very_simple_size, node_component_dynamics_statistics_bars, node_component_hot_halo_cold_mode, node_component_hot_halo_outflow_tracking, node_component_hot_halo_standard

node_component_hot_halo_very_simple node_component_inter_output_standard
node_component_mass_flow_statistics_-standard node_component_merging_statistics_-recent
node_component_merging_statistics_-standard node_component_position_preset_-orphans
node_component_position_trace_dark_-matter node_component_satellite_orbiting
node_component_satellite_standard node_component_satellite_very_simple
node_component_spheroid_standard node_component_spheroid_very_simple
node_component_spin_random node_component_spin_vitvitska

subroutine: node_components_thread_uninitialize
Description: Perform per-thread uninitialization tasks for node components.
Code lines: 85
Contained by: module node_components
Modules used: node_component_basic_standard_-extended node_component_black_hole_noncentral
node_component_black_hole_simple node_component_black_hole_standard
node_component_dark_matter_profile_-scale node_component_dark_matter_profile_-scale_shape
node_component_disk_standard node_component_disk_very_simple
node_component_disk_very_simple_size node_component_dynamics_statistics_-bars
node_component_hot_halo_cold_mode node_component_hot_halo_outflow_-tracking
node_component_hot_halo_standard node_component_hot_halo_very_simple
node_component_inter_output_standard node_component_mass_flow_statistics_-standard
node_component_merging_statistics_-recent node_component_merging_statistics_-standard
node_component_position_preset_-orphans node_component_position_trace_dark_-matter
node_component_satellite_orbiting node_component_satellite_standard
node_component_satellite_very_simple node_component_spheroid_standard
node_component_spheroid_very_simple node_component_spin_random
node_component_spin_vitvitska

subroutine: node_components_uninitialize
Description: Perform uninitialization tasks for node components.
Code lines: 6
Contained by: module node_components

file: objects.nodes.components.age_statistics.standard.F90
Description: Contains a module which implements the standard galaxy age statistics component.
Code lines: 264

module: node_component_age_statistics_standard
Description: Implements the standard galaxy age statistics component.

Code lines: 242
Contained by: file objects.nodes.components.age_statistics.standard.F90
Used by: subroutine satellitemergerprocess; subroutine standardderivativescompute; subroutine standardevolve; subroutine node_components_initialize

subroutine: node_component_age_statistics_standard_inactive
Description: Set Jacobian zero status for properties of node.
Code lines: 21
Contained by: module node_component_age_statistics_standard
Modules used: galacticus_nodes; stellar_luminosities_structure

subroutine: node_component_age_statistics_standard_initialize
Description: Initializes the tree node standard disk methods module.
Code lines: 18
Contained by: module node_component_age_statistics_standard
Modules used: galacticus_nodes; input_parameters

subroutine: node_component_age_statistics_standard_rate_compute
Description: Compute the exponential disk node mass rate of change.
Code lines: 44
Contained by: module node_component_age_statistics_standard
Modules used: galacticus_nodes; output_times

subroutine: node_component_age_statistics_standard_satellite_merging
Description: Remove any age statistics quantities associated with node and add them to the merge target.
Code lines: 54
Contained by: module node_component_age_statistics_standard
Modules used: galacticus_error; galacticus_nodes; satellite_merging_mass_movements; satellite_merging_remnant_properties

subroutine: node_component_age_statistics_standard_scale_set
Description: Set scales for properties of node.
Code lines: 28
Contained by: module node_component_age_statistics_standard
Modules used: galacticus_nodes

file: objects.nodes.components.basic.extended.F90
Description: Contains a module which extends the standard implementation of basic component to track the Bertschinger mass.
Code lines: 443

module: node_component_basic_standard_extended
Description: Extends the standard implementation of basic component to track the Bertschinger mass.
Code lines: 420
Contained by: file objects.nodes.components.basic.extended.F90
Modules used: cosmology_functions; cosmology_parameters; virial_density_contrast
Used by: subroutine merger_tree_initialize; subroutine standardderivativescompute

subroutine standardevolve
subroutine standardmerge
subroutine standardpromote
subroutine node_components_initialize
subroutine node_components_thread_initialize
subroutine node_components_thread_uninitialize

subroutine: node_component_basic_extended_bertschinger_solver
Description: Compute the Bertschinger mass and turnaround radii
Code lines: 38
Contained by: module node_component_basic_standard_extended
Modules used: dark_matter_profile_mass_definitions galacticus_nodes

subroutine: node_component_basic_extended_bindings
Description: Initializes the "extended" implementation of the basic component.
Code lines: 46
Contained by: module node_component_basic_standard_extended
Modules used: galacticus_nodes input_parameters
iso_varying_string spherical_collapse_matter_dark_energy

function: node_component_basic_extended_mass_bertschinger
Description: Return the Bertschinger mass.
Code lines: 9
Contained by: module node_component_basic_standard_extended
Modules used: galacticus_nodes

function: node_component_basic_extended_radius_turnaround
Description: Return the turnaround radius.
Code lines: 9
Contained by: module node_component_basic_standard_extended
Modules used: galacticus_nodes

subroutine: node_component_basic_extended_thread_initialize
Description: Initializes the tree node random spin module.
Code lines: 12
Contained by: module node_component_basic_standard_extended
Modules used: galacticus_nodes input_parameters

subroutine: node_component_basic_extended_thread_uninitialize
Description: Uninitializes the tree node random spin module.
Code lines: 10
Contained by: module node_component_basic_standard_extended
Modules used: galacticus_nodes

subroutine: node_component_basic_standard_extended_initialize
Description: Set the mass accretion rate for node.
Code lines: 68
Contained by: module node_component_basic_standard_extended
Modules used: galacticus_nodes

subroutine: `node_component_basic_standard_extended_node_merger`
Description: Switch off accretion of new mass onto this node once it becomes a satellite.
Code lines: 17
Contained by: module node_component_basic_standard_extended
Modules used: galacticus_nodes

subroutine: `node_component_basic_standard_extended_promote`
Description: Ensure that `node` is ready for promotion to its parent. In this case, we simply update the mass of `node` to be that of its parent.
Code lines: 30
Contained by: module node_component_basic_standard_extended
Modules used: galacticus_error galacticus_nodes

subroutine: `node_component_basic_standard_extended_rate_compute`
Description: Compute rates of change of properties in the standard implementation of the basic component.
Code lines: 25
Contained by: module node_component_basic_standard_extended
Modules used: galacticus_nodes

subroutine: `node_component_basic_standard_extended_scale_set`
Description: Set scales for properties in the standard implementation of the basic component.
Code lines: 20
Contained by: module node_component_basic_standard_extended
Modules used: galacticus_nodes

function: `node_component_basic_standard_extended_unresolved_mass`
Description: Return the unresolved mass for `node`.
Code lines: 20
Contained by: module node_component_basic_standard_extended
Modules used: galacticus_nodes

file: `objects.nodes.components.basic.extended_tracking.F90`
Description: Contains a module which extends the extended implementation of basic component to track the maximum progenitor mass.
Code lines: 105

module: `node_component_basic_extended_tracking`
Description: Extends the extended implementation of basic component to track the maximum progenitor mass.
Code lines: 83
Contained by: file objects.nodes.components.basic.extended_tracking.F90
Used by: subroutine merger_tree_initialize subroutine standardpromote

subroutine: `node_component_basic_extended_tracking_promote`
Description: Ensure that `node` is ready for promotion to its parent. In this case, we simply update the maximum mass of `node` to be that of its parent.
Code lines: 22
Contained by: module node_component_basic_extended_tracking

Modules used: galacticus_error galacticus_nodes

subroutine: node_component_basic_extended_tree_tracking_initialize
Description: Set the mass accretion rate for node.
Code lines: 25
Contained by: module node_component_basic_extended_tracking
Modules used: galacticus_nodes

file: objects.nodes.components.basic.non_evolving.F90
Description: Contains a module with the standard implementation of basic tree node methods.
Code lines: 142

module: node_component_basic_non_evolving
Description: A non-evolving implementation of basic tree node methods.
Code lines: 120
Contained by: file objects.nodes.components.basic.non_evolving.F90
Used by: subroutine standardderivativescompute subroutine standardevolve subroutine standardpromote

subroutine: node_component_basic_non_evolving_promote
Description: Ensure that node is ready for promotion to its parent. In this case, we simply update the mass of node to be that of its parent.
Code lines: 24
Contained by: module node_component_basic_non_evolving
Modules used: galacticus_error galacticus_nodes

subroutine: node_component_basic_non_evolving_rate_compute
Description: Compute rates of change of properties in the standard implementation of the basic component.
Code lines: 26
Contained by: module node_component_basic_non_evolving
Modules used: galacticus_nodes

subroutine: node_component_basic_non_evolving_scale_set
Description: Set scales for properties in the standard implementation of the basic component.
Code lines: 17
Contained by: module node_component_basic_non_evolving
Modules used: galacticus_nodes

file: objects.nodes.components.basic.standard.F90
Description: Contains a module with the standard implementation of basic tree node methods.
Code lines: 339

module: node_component_basic_standard
Description: The standard implementation of basic tree node methods.
Code lines: 317
Contained by: file objects.nodes.components.basic.standard.F90
Used by: subroutine equilibriumsolve subroutine fixedsolve subroutine linearsolve subroutine simplesolve

subroutine merger_tree_initialize
subroutine standardderivativescompute
subroutine standardevolve
subroutine standardmerge
subroutine standardpoststepprocessing
subroutine standardpromote

subroutine: node_component_basic_standard_plausibility
Description: Determines whether the basic is physically plausible. Require the mass and time to be positive.
Code lines: 16
Contained by: module node_component_basic_standard
Modules used: galacticus_nodes

subroutine: node_component_basic_standard_post_step
Description: Trim histories attached to the disk.
Code lines: 18
Contained by: module node_component_basic_standard
Modules used: fgsl galacticus_nodes

subroutine: node_component_basic_standard_promote
Description: Ensure that node is ready for promotion to its parent. In this case, we simply update the mass of node to be that of its parent.
Code lines: 38
Contained by: module node_component_basic_standard
Modules used: galacticus_error galacticus_nodes
iso_varying_string string_handling

subroutine: node_component_basic_standard_rate_compute
Description: Compute rates of change of properties in the standard implementation of the basic component.
Code lines: 25
Contained by: module node_component_basic_standard
Modules used: galacticus_nodes

subroutine: node_component_basic_standard_scale_set
Description: Set scales for properties in the standard implementation of the basic component.
Code lines: 20
Contained by: module node_component_basic_standard
Modules used: galacticus_nodes

subroutine: node_component_basic_standard_stop_accretion
Description: Switch off accretion of new mass onto this node once it becomes a satellite.
Code lines: 18
Contained by: module node_component_basic_standard
Modules used: galacticus_nodes

subroutine: node_component_basic_standard_tree_initialize
Description: Set the mass accretion rate for node.
Code lines: 75
Contained by: module node_component_basic_standard
Modules used: galacticus_nodes

function: `node_component_basic_standard_unresolved_mass`
Description: Return the unresolved mass for `node`.
Code lines: 20
Contained by: module node_component_basic_standard
Modules used: galacticus_nodes

file: `objects.nodes.components.basic.standard_tracking.F90`
Description: Contains a module which extends the standard implementation of basic component to track the maximum progenitor mass.
Code lines: 105

module: `node_component_basic_standard_tracking`
Description: Extends the standard implementation of basic component to track the maximum progenitor mass.
Code lines: 83
Contained by: file objects.nodes.components.basic.standard_tracking.F90
Used by: subroutine merger_tree_initialize, subroutine standardpromote

subroutine: `node_component_basic_standard_tracking_promote`
Description: Ensure that `node` is ready for promotion to its parent. In this case, we simply update the maximum mass of `node` to be that of its parent.
Code lines: 22
Contained by: module node_component_basic_standard_tracking
Modules used: galacticus_error, galacticus_nodes

subroutine: `node_component_basic_standard_tree_tracking_initialize`
Description: Set the mass accretion rate for `node`.
Code lines: 25
Contained by: module node_component_basic_standard_tracking
Modules used: galacticus_nodes

file: `objects.nodes.components.black_hole.non_central.F90`
Description: Contains a module which implements the standard black hole node component.
Code lines: 390

module: `node_component_black_hole_noncentral`
Description: Implement black hole tree node methods.
Code lines: 368
Contained by: file objects.nodes.components.black_hole.non_central.F90
Modules used: black_hole_binary_mergers, black_hole_binary_recoil_velocities, black_hole_binary_separations, dark_matter_halo_scales
Used by: subroutine standardderivativescompute, subroutine standardevolve, subroutine node_components_initialize, subroutine node_components_thread_initialize, subroutine node_components_thread_uninitialize

subroutine: `node_component_black_hole_noncentral_initialize`
Description: Initializes the noncentral black hole component module.
Code lines: 16
Contained by: module `node_component_black_hole_noncentral`
Modules used: `input_parameters`

subroutine: `node_component_black_hole_noncentral_merge_black_holes`
Description: Merge two black holes.
Code lines: 38
Contained by: module `node_component_black_hole_noncentral`
Modules used: `galacticus_nodes`

subroutine: `node_component_black_hole_noncentral_rate_compute`
Description: Compute the black hole node mass rate of change.
Code lines: 82
Contained by: module `node_component_black_hole_noncentral`
Modules used: `galacticus_nodes` `numerical_constants_astronomical`

function: `node_component_black_hole_noncentral_recoil_escapes`
Description: Return true if the given recoil velocity is sufficient to eject a black hole from the halo.
Code lines: 26
Contained by: module `node_component_black_hole_noncentral`
Modules used: `galactic_structure_options` `galactic_structure_potentials` `galacticus_nodes`

subroutine: `node_component_black_hole_noncentral_scale_set`
Description: Set scales for properties of `node`.
Code lines: 24
Contained by: module `node_component_black_hole_noncentral`
Modules used: `galacticus_nodes`

subroutine: `node_component_black_hole_noncentral_thread_initialize`
Description: Initializes the tree node random spin module.
Code lines: 14
Contained by: module `node_component_black_hole_noncentral`
Modules used: `galacticus_nodes` `input_parameters`

subroutine: `node_component_black_hole_noncentral_thread_uninitialize`
Description: Uninitializes the tree node random spin module.
Code lines: 12
Contained by: module `node_component_black_hole_noncentral`
Modules used: `galacticus_nodes`

subroutine: `node_component_black_hole_noncentral_triple_interaction`
Description: Handles triple black holes interactions, using conditions similar to those of Volonteri et al. [2003].
Code lines: 76
Contained by: module `node_component_black_hole_noncentral`

Modules used: galacticus_nodes numerical_constants_physical

file: objects.nodes.components.black_hole.simple.F90
Description: Contains a module which implements the simple black hole node component.
Code lines: 433

module: node_component_black_hole_simple
Description: Implements the simple black hole node component.
Code lines: 411
Contained by: file objects.nodes.components.black_hole.simple.F90
Modules used: black_hole_binary_mergers cooling_radii dark_matter_halo_scales
Used by: subroutine satellitemergerprocess; subroutine standardderivativescompute; subroutine standardevolve; subroutine standardoutput; subroutine standardpropertiescount; subroutine standardpropertynamesestablish; subroutine node_components_initialize; subroutine node_components_thread_initialize; subroutine node_components_thread_uninitialize

subroutine: node_component_black_hole_simple_create
Description: Creates a simple black hole component for thisNode.
Code lines: 12
Contained by: module node_component_black_hole_simple
Modules used: galacticus_nodes

subroutine: node_component_black_hole_simple_initialize
Description: Initializes the simple black hole node component module.
Code lines: 63
Contained by: module node_component_black_hole_simple
Modules used: galacticus_nodes input_parameters

function: node_component_black_hole_simple_matches
Description: Return true if the black hole component of thisNode is a match to the simple implementation.
Code lines: 18
Contained by: module node_component_black_hole_simple
Modules used: galacticus_nodes

subroutine: node_component_black_hole_simple_output
Description: Store black hole properties in the Galacticus output file buffers.
Code lines: 31
Contained by: module node_component_black_hole_simple
Modules used: galacticus_nodes kind_numbers multi_counters

subroutine: node_component_black_hole_simple_output_count
Description: Account for the number of black hole properties to be written to the the Galacticus output file.

Code lines: 15
Contained by: module node_component_black_hole_simple
Modules used: galacticus_nodes

subroutine: node_component_black_hole_simple_output_names
Description: Set names of black hole properties to be written to the Galacticus output file.
Code lines: 22
Contained by: module node_component_black_hole_simple
Modules used: galacticus_nodes numerical_constants_astronomical

subroutine: node_component_black_hole_simple_rate_compute
Description: Compute the black hole mass rate of change.
Code lines: 78
Contained by: module node_component_black_hole_simple
Modules used: galacticus_nodes numerical_constants_physical

subroutine: node_component_black_hole_simple_satellite_merging
Description: Merge (instantaneously) any simple black hole associated with thisNode before it merges with its host halo.
Code lines: 23
Contained by: module node_component_black_hole_simple
Modules used: galacticus_nodes

subroutine: node_component_black_hole_simple_scale_set
Description: Set scales for properties of thisNode.
Code lines: 19
Contained by: module node_component_black_hole_simple
Modules used: galacticus_nodes

subroutine: node_component_black_hole_simple_thread_initialize
Description: Initializes the tree node random spin module.
Code lines: 13
Contained by: module node_component_black_hole_simple
Modules used: galacticus_nodes input_parameters

subroutine: node_component_black_hole_simple_thread_uninitialize
Description: Uninitializes the tree node random spin module.
Code lines: 11
Contained by: module node_component_black_hole_simple
Modules used: galacticus_nodes

file: objects.nodes.components.black_hole.simple.structure.F90
Description: Contains a module which implements the structure tasks for the simple black hole node component.
Code lines: 137

module: node_component_black_hole_simple_structure
Description: Implements the structure tasks for the simple black hole node component.
Code lines: 115

Contained by: file objects.nodes.components.black_hole.simple.structure.F90
Used by: subroutine adiabaticgnedin2004computefactors; function galactic_structure_potential; function galactic_structure_rotation_curve; function galactic_structure_rotation_curve_gradient

function: node_component_black_hole_simple_potential
Description: Compute the gravitational potential due to a black hole.
Code lines: 29
Contained by: module node_component_black_hole_simple_structure
Modules used: black_hole_fundamentals; galactic_structure_options; galacticus_nodes; numerical_constants_physical

function: node_component_black_hole_simple_rotation_curve
Description: Computes the rotation curve for the central black hole. Assumes a point mass black hole with a Keplerian rotation curve, *except* that the rotation speed is limited to never exceed the speed of light.
Code lines: 31
Contained by: module node_component_black_hole_simple_structure
Modules used: black_hole_fundamentals; galactic_structure_options; galacticus_nodes; numerical_constants_physical

function: node_component_black_hole_simple_rotation_curve_gradient
Description: Computes the rotation curve gradient for the central black hole. Assumes a point mass black hole with a Keplerian rotation curve, *except* that the rotation speed is limited to never exceed the speed of light.
Code lines: 32
Contained by: module node_component_black_hole_simple_structure
Modules used: black_hole_fundamentals; galactic_structure_options; galacticus_nodes; numerical_constants_physical

file: objects.nodes.components.black_hole.standard.F90
Description: Contains a module which implements the standard black hole node component.
Code lines: 1031

module: node_component_black_hole_standard
Description: Implement black hole tree node methods.
Code lines: 1009
Contained by: file objects.nodes.components.black_hole.standard.F90
Modules used: accretion_disks; black_hole_binary_initial_separation; black_hole_binary_mergers; black_hole_binary_recoil_velocities; black_hole_binary_separations; cooling_radii; cosmology_parameters; dark_matter_halo_scales; hot_halo_temperature_profiles
Used by: subroutine satellitemergerprocess; subroutine standardderivativescompute; subroutine standardevolve; subroutine standardpoststepprocessing; subroutine standardoutput; subroutine standardpropertiescount; subroutine standardpropertynamesestablish; subroutine node_components_initialize

subroutine node_components_thread_initialize
subroutine node_components_thread_uninitialize
subroutine evolveforestsperform

function: hot_mode_fraction
Description: A simple interpolating function which is used as a measure of the fraction of a halo which is in the hot accretion mode.
Code lines: 19
Contained by: module node_component_black_hole_standard
Modules used: galacticus_nodes

function: node_component_black_hole_standard_accretion_rate
Description: Return the rest mass accretion rate onto a standard black hole.
Code lines: 10
Contained by: module node_component_black_hole_standard
Modules used: galacticus_nodes

subroutine: node_component_black_hole_standard_create
Description: Creates a black hole component for node.
Code lines: 14
Contained by: module node_component_black_hole_standard
Modules used: galacticus_nodes

subroutine: node_component_black_hole_standard_initialize
Description: Initializes the standard black hole component module.
Code lines: 126
Contained by: module node_component_black_hole_standard
Modules used: galacticus_nodes input_parameters

subroutine: node_component_black_hole_standard_mass_accretion_rate
Description: Returns the rate of mass accretion onto the black hole in node.
Code lines: 112
Contained by: module node_component_black_hole_standard
Modules used: black_hole_fundamentals bondi_hoyle_lyttleton_accretion galactic_structure_densities galactic_structure_options galacticus_nodes ideal_gases_thermodynamics numerical_constants_astronomical

function: node_component_black_hole_standard_matches
Description: Return true if the black hole component of node is a match to the standard implementation.
Code lines: 18
Contained by: module node_component_black_hole_standard
Modules used: galacticus_nodes

subroutine: node_component_black_hole_standard_output
Description: Store black hole properties in the Galacticus output file buffers.
Code lines: 36
Contained by: module node_component_black_hole_standard
Modules used: galacticus_nodes kind_numbers

multi_counters

subroutine: node_component_black_hole_standard_output_count
Description: Account for the number of black hole properties to be written to the the Galacticus output file.
Code lines: 15
Contained by: module node_component_black_hole_standard
Modules used: galacticus_nodes

subroutine: node_component_black_hole_standard_output_merger
Description: Outputs properties of merging black holes.
Code lines: 31
Contained by: module node_component_black_hole_standard
Modules used: galacticus_hdf5 galacticus_nodes

subroutine: node_component_black_hole_standard_output_names
Description: Set names of black hole properties to be written to the Galacticus output file.
Code lines: 33
Contained by: module node_component_black_hole_standard
Modules used: galacticus_nodes numerical_constants_astronomical

subroutine: node_component_black_hole_standard_output_properties
Description: Output properties for all black holes in node.
Code lines: 88
Contained by: module node_component_black_hole_standard
Modules used: galacticus_hdf5 galacticus_nodes
iso_c_binding iso_varying_string
kind_numbers memory_management
string_handling

subroutine: node_component_black_hole_standard_post_evolve
Description: Keep black hole spin in physical range.
Code lines: 30
Contained by: module node_component_black_hole_standard
Modules used: fgsl galacticus_nodes

function: node_component_black_hole_standard_radiative_efficiency
Description: Return the radiative efficiency of a standard black hole.
Code lines: 8
Contained by: module node_component_black_hole_standard
Modules used: galacticus_nodes

subroutine: node_component_black_hole_standard_rate_compute
Description: Compute the black hole node mass rate of change.
Code lines: 116
Contained by: module node_component_black_hole_standard
Modules used: black_hole_binary_separations galacticus_nodes
numerical_constants_astronomical numerical_constants_physical

function: node_component_black_hole_standard_recoil_escapes
Description: Return true if the given recoil velocity is sufficient to eject a black hole from the halo.
Code lines: 26
Contained by: module node_component_black_hole_standard
Modules used: galactic_structure_options galactic_structure_potentials galacticus_nodes

subroutine: node_component_black_hole_standard_satellite_merging
Description: Merge any black hole associated with node before it merges with its host halo.
Code lines: 68
Contained by: module node_component_black_hole_standard
Modules used: galacticus_nodes

subroutine: node_component_black_hole_standard_scale_set
Description: Set scales for properties of node.
Code lines: 32
Contained by: module node_component_black_hole_standard
Modules used: galacticus_nodes

subroutine: node_component_black_hole_standard_thread_initialize
Description: Initializes the tree node standard black hole module.
Code lines: 19
Contained by: module node_component_black_hole_standard
Modules used: galacticus_nodes input_parameters

subroutine: node_component_black_hole_standard_thread_uninitialize
Description: Uninitializes the tree node standard black hole module.
Code lines: 17
Contained by: module node_component_black_hole_standard
Modules used: galacticus_nodes

file: objects.nodes.components.black_hole.standard.structure_tasks.F90
Description: Contains a module which implements galactic structure tasks for the standard black hole node component.
Code lines: 138

module: node_component_black_hole_standard_structure_tasks
Description: Implements galactic structure tasks for the standard black hole tree node component.
Code lines: 114
Contained by: file objects.nodes.components.black_hole.standard.structure_tasks.F90
Used by: subroutine adiabaticgnedin2004computefactors function galactic_structure_potential function galactic_structure_rotation_curve function galactic_structure_rotation_curve_gradient

function: node_component_black_hole_standard_potential
Description: Compute the gravitational potential due to a black hole.
Code lines: 28

Contained by: module node_component_black_hole_standard_structure_tasks
Modules used: black_hole_fundamentals, galactic_structure_options, galacticus_nodes, numerical_constants_physical

function: node_component_black_hole_standard_rotation_curve
Description: Computes the rotation curve for the central black hole. Assumes a point mass black hole with a Keplerian rotation curve, *except* that the rotation speed is limited to never exceed the speed of light.
Code lines: 31
Contained by: module node_component_black_hole_standard_structure_tasks
Modules used: black_hole_fundamentals, galactic_structure_options, galacticus_nodes, numerical_constants_physical

function: node_component_black_hole_standard_rotation_curve_gradient
Description: Computes the rotation curve gradient for the central black hole. Assumes a point mass black hole with a Keplerian rotation curve, *except* that the rotation speed is limited to never exceed the speed of light.
Code lines: 32
Contained by: module node_component_black_hole_standard_structure_tasks
Modules used: black_hole_fundamentals, galactic_structure_options, galacticus_nodes, numerical_constants_physical

file: objects.nodes.components.dark_matter_profile.scale.F90
Description: Contains a module which implements a dark matter profile method that provides a scale radius.
Code lines: 378

module: node_component_dark_matter_profile_scale
Description: Implements a dark matter profile method that provides a scale radius.
Code lines: 356
Contained by: file objects.nodes.components.dark_matter_profile.scale.F90
Modules used: dark_matter_halo_scales, dark_matter_profile_scales, galacticus_nodes
Used by: subroutine equilibriumsolve; subroutine fixedsolve; subroutine linearsolve; subroutine simplesolve; subroutine halo_model_projected_-correlation; subroutine merger_tree_initialize; subroutine standardderivativescompute; subroutine standardevolve; subroutine standardpromote; subroutine merger_tree_structure_output; function projectedcorrelationfunctionconstructorinternal; subroutine node_components_initialize; subroutine node_components_thread_-initialize; subroutine node_components_thread_-uninitialize

subroutine: node_component_dark_matter_profile_scale_initialize
Description: Initializes the "scale" implementation of the dark matter halo profile component.
Code lines: 35
Contained by: module node_component_dark_matter_profile_scale
Modules used: input_parameters

subroutine: `node_component_dark_matter_profile_scale_plausibility`
Description: Determines whether the dark matter profile is physically plausible for radius solving tasks.
Code lines: 17
Contained by: module `node_component_dark_matter_profile_scale`
Modules used: `galacticus_nodes`

subroutine: `node_component_dark_matter_profile_scale_promote`
Description: Ensure that `node` is ready for promotion to its parent. In this case, we simply update the growth rate of `node` to be that of its parent.
Code lines: 25
Contained by: module `node_component_dark_matter_profile_scale`
Modules used: `galacticus_error` `galacticus_nodes`

subroutine: `node_component_dark_matter_profile_scale_rate_compute`
Description: Compute the rate of change of the scale radius.
Code lines: 29
Contained by: module `node_component_dark_matter_profile_scale`
Modules used: `galacticus_nodes`

function: `node_component_dark_matter_profile_scale_scale`
Description: Return the scale radius in the dark matter halo profile.
Code lines: 20
Contained by: module `node_component_dark_matter_profile_scale`
Modules used: `galacticus_nodes`

subroutine: `node_component_dark_matter_profile_scale_scale_set`
Description: Set scales for properties of `node`.
Code lines: 16
Contained by: module `node_component_dark_matter_profile_scale`
Modules used: `galacticus_nodes`

subroutine: `node_component_dark_matter_profile_scale_thread_initialize`
Description: Initializes the tree node scale dark matter profile module.
Code lines: 12
Contained by: module `node_component_dark_matter_profile_scale`
Modules used: `galacticus_nodes` `input_parameters`

subroutine: `node_component_dark_matter_profile_scale_thread_uninitialize`
Description: Uninitializes the tree node scale dark matter profile module.
Code lines: 10
Contained by: module `node_component_dark_matter_profile_scale`
Modules used: `galacticus_nodes`

subroutine: `node_component_dark_matter_profile_scale_tree_initialize`
Description: Initialize the scale radius of `node`.
Code lines: 52
Contained by: module `node_component_dark_matter_profile_scale`

Modules used: galacticus_error galacticus_nodes merger_tree_walkers

subroutine: node_component_dark_matter_profile_scale_tree_output
Description: Write the scale radius property to a full merger tree output.
Code lines: 37
Contained by: module node_component_dark_matter_profile_scale
Modules used: galacticus_nodes io_hdf5 merger_tree_walkers numerical_constants_astronomical

file: objects.nodes.components.dark_matter_profile.scale_preset.F90
Description: Contains a module which implements a dark matter profile method that provides a scale radius.
Code lines: 169

module: node_component_dark_matter_profile_scale_preset
Description: Implements a dark matter profile method that provides a scale radius.
Code lines: 147
Contained by: file objects.nodes.components.dark_matter_profile.scale_preset.F90
Used by: subroutine merger_tree_initialize subroutine standardderivativescompute subroutine standardevolve subroutine standardpromote

subroutine: node_component_dark_matter_profile_scale_preset_promote
Description: Ensure that node is ready for promotion to its parent. In this case, we simply update the growth rate of node to be that of its parent.
Code lines: 25
Contained by: module node_component_dark_matter_profile_scale_preset
Modules used: galacticus_error galacticus_nodes

subroutine: node_component_dark_matter_profile_scale_preset_rate_compute
Description: Compute the rate of change of the scale radius.
Code lines: 25
Contained by: module node_component_dark_matter_profile_scale_preset
Modules used: dark_matter_halo_scales galacticus_nodes

subroutine: node_component_dark_matter_profile_scale_preset_scale_set
Description: Set scales for properties of node.
Code lines: 16
Contained by: module node_component_dark_matter_profile_scale_preset
Modules used: galacticus_nodes

subroutine: node_component_dark_matter_profile_scale_preset_tree_initialize
Description: Initialize the scale radius of node.
Code lines: 30
Contained by: module node_component_dark_matter_profile_scale_preset
Modules used: galacticus_nodes

file: objects.nodes.components.dark_matter_profile.scale_shape.F90
Description: Contains a module which implements a dark matter profile method that provides a scale radius and a shape parameter.

Code lines: 295

module: `node_component_dark_matter_profile_scale_shape`

Description: Implements a dark matter profile method that provides a scale radius and a shape parameter.
Code lines: 273
Contained by: file objects.nodes.components.dark_matter_profile.scale_shape.F90
Modules used: dark_matter_profiles_shape, galacticus_nodes
Used by: subroutine merger_tree_initialize, subroutine standardderivativescompute, subroutine standardevolve, subroutine standardpromote, subroutine merger_tree_structure_output, subroutine node_components_initialize, subroutine node_components_thread_initialize, subroutine node_components_thread_uninitialize

subroutine: `node_component_dark_matter_profile_scale_shape_initialize`

Description: Initializes the "scale" implementation of the dark matter halo profile component.
Code lines: 18
Contained by: module node_component_dark_matter_profile_scale_shape
Modules used: input_parameters

subroutine: `node_component_dark_matter_profile_scale_shape_promote`

Description: Ensure that `node` is ready for promotion to its parent. In this case, we simply update the growth rate of `node` to be that of its parent.
Code lines: 25
Contained by: module node_component_dark_matter_profile_scale_shape
Modules used: galacticus_error, galacticus_nodes

subroutine: `node_component_dark_matter_profile_scale_shape_rate_compute`

Description: Compute the rate of change of the scale radius.
Code lines: 24
Contained by: module node_component_dark_matter_profile_scale_shape
Modules used: galacticus_nodes

subroutine: `node_component_dark_matter_profile_scale_shape_scale_set`

Description: Set scales for properties of `node`.
Code lines: 16
Contained by: module node_component_dark_matter_profile_scale_shape
Modules used: galacticus_nodes

function: `node_component_dark_matter_profile_scale_shape_shape`

Description: Return the shape parameter in the dark matter halo profile.
Code lines: 15
Contained by: module node_component_dark_matter_profile_scale_shape

subroutine: `node_component_dark_matter_profile_scale_shape_thread_init`

Description: Initializes the tree node random spin module.
Code lines: 11
Contained by: module node_component_dark_matter_profile_scale_shape
Modules used: galacticus_nodes, input_parameters

subroutine: `node_component_dark_matter_profile_scale_shape_thread_uninit`

Description: Uninitializes the tree node random spin module.
Code lines: 9
Contained by: module `node_component_dark_matter_profile_scale_shape`
Modules used: `galacticus_nodes`

subroutine: `node_component_dark_matter_profile_scale_shape_tree_initialize`

Description: Initialize the scale radius of `node`.
Code lines: 33
Contained by: module `node_component_dark_matter_profile_scale_shape`
Modules used: `galacticus_nodes`

subroutine: `node_component_dark_matter_profile_scale_shape_tree_output`

Description: Write the scale radius property to a full merger tree output.
Code lines: 33
Contained by: module `node_component_dark_matter_profile_scale_shape`
Modules used: `galacticus_nodes` `io_hdf5` `merger_tree_walkers`

file: `objects.nodes.components.disk.standard.F90`

Description: Contains a module which implements the standard disk node component.
Code lines: 1385

module: `node_component_disk_standard`

Description: Implements the standard disk node component.
Code lines: 1363
Contained by: file `objects.nodes.components.disk.standard.F90`
Modules used: `dark_matter_halo_scales` `galactic_dynamics_bar_instabilities` `iso_varying_string` `ram_pressure_stripping_mass_loss_rate_disks` `star_formation_feedback_disks` `star_formation_feedback_expulsion_disks` `star_formation_histories` `star_formation_timescales_disks` `stellar_population_properties` `tidal_stripping_mass_loss_rate_disks`
Used by: subroutine `equilibriumsolve` subroutine `fixedsolve` subroutine `linearsolve` subroutine `simplesolve` subroutine `galacticus_calculations_reset` subroutine `galacticus_state_retrieve` subroutine `galacticus_state_store` subroutine `satellitemergerprocess` subroutine `standardderivativescompute` subroutine `standardevolve` subroutine `standardfinalstateprocessing` subroutine `standardpoststepprocessing` subroutine `node_components_initialize` subroutine `node_components_thread_initialize` subroutine `node_components_thread_uninitialize` subroutine `evolveforestsperform`

subroutine: `node_component_disk_standard_calculation_reset`
Description: Reset standard disk structure calculations.
Code lines: 9
Contained by: module `node_component_disk_standard`
Modules used: `galacticus_nodes` `node_component_disk_standard_data`

subroutine: `node_component_disk_standard_create`
Description: Create properties in an standard disk component.
Code lines: 46
Contained by: module `node_component_disk_standard`
Modules used: `galacticus_nodes` `histories`

subroutine: `node_component_disk_standard_final_state`
Description: Record the final state of the disk at the end of the timestep prior to begin evaluation of integrals for inactive properties.
Code lines: 19
Contained by: module `node_component_disk_standard`
Modules used: `galacticus_nodes`

subroutine: `node_component_disk_standard_inactive`
Description: Set Jacobian zero status for properties of `node`.
Code lines: 16
Contained by: module `node_component_disk_standard`
Modules used: `galacticus_nodes` `stellar_luminosities_structure`

subroutine: `node_component_disk_standard_initialize`
Description: Initializes the tree node standard disk methods module.
Code lines: 80
Contained by: module `node_component_disk_standard`
Modules used: `abundances_structure` `galacticus_error` `galacticus_nodes` `input_parameters` `node_component_disk_standard_data`

subroutine: `node_component_disk_standard_post_evolve`
Description: Trim histories attached to the disk.
Code lines: 23
Contained by: module `node_component_disk_standard`
Modules used: `galacticus_nodes` `histories` `stellar_luminosities_structure`

subroutine: `node_component_disk_standard_post_step`
Description: Trim histories attached to the disk.
Code lines: 138
Contained by: module `node_component_disk_standard`
Modules used: `abundances_structure` `fgsl` `galacticus_display` `galacticus_error` `galacticus_nodes` `iso_varying_string`

stellar_luminosities_structure string_handling

subroutine: node_component_disk_standard_pre_evolve
Description: Ensure the disk has been initialized.
Code lines: 20
Contained by: module node_component_disk_standard
Modules used: galacticus_nodes

function: node_component_disk_standard_radius_solve
Description: Return the radius of the standard disk used in structure solvers.
Code lines: 10
Contained by: module node_component_disk_standard
Modules used: galacticus_nodes

subroutine: node_component_disk_standard_radius_solve_set
Description: Set the radius of the standard disk used in structure solvers.
Code lines: 11
Contained by: module node_component_disk_standard
Modules used: galacticus_nodes

subroutine: node_component_disk_standard_radius_solver
Description: Interface for the size solver algorithm.
Code lines: 53
Contained by: module node_component_disk_standard
Modules used: galacticus_nodes node_component_disk_standard_data numerical_constants_physical tables

subroutine: node_component_disk_standard_radius_solver_plausibility
Description: Determines whether the disk is physically plausible for radius solving tasks. Require that it have non-zero mass and angular momentum.
Code lines: 37
Contained by: module node_component_disk_standard
Modules used: galacticus_nodes

subroutine: node_component_disk_standard_rate_compute
Description: Compute the standard disk node mass rate of change.
Code lines: 278
Contained by: module node_component_disk_standard
Modules used: abundances_structure galactic_structure_options galacticus_error galacticus_nodes histories numerical_constants_astronomical stellar_luminosities_structure

subroutine: node_component_disk_standard_satellite_merging
Description: Transfer any standard disk associated with node to its host halo.
Code lines: 101
Contained by: module node_component_disk_standard
Modules used: abundances_structure galacticus_error galacticus_nodes histories

satellite_merging_mass_movements satellite_merging_remnant_properties
stellar_luminosities_structure

subroutine: node_component_disk_standard_scale_set
Description: Set scales for properties of node.
Code lines: 63
Contained by: module node_component_disk_standard
Modules used: abundances_structure galacticus_nodes
histories stellar_luminosities_structure

subroutine: node_component_disk_standard_star_formation_history_output
Description: Store the star formation history in the output file.
Code lines: 23
Contained by: module node_component_disk_standard
Modules used: galacticus_nodes histories
iso_c_binding kind_numbers

function: node_component_disk_standard_star_formation_rate
Description: Return the star formation rate of the standard disk.
Code lines: 26
Contained by: module node_component_disk_standard
Modules used: galacticus_nodes

subroutine: node_component_disk_standard_state_retrieve
Description: Retrieve the tabulation state from the file.
Code lines: 21
Contained by: module node_component_disk_standard
Modules used: fgsl galacticus_display
galacticus_error iso_c_binding
node_component_disk_standard_data

subroutine: node_component_disk_standard_state_store
Description: Write the tablulation state to file.
Code lines: 16
Contained by: module node_component_disk_standard
Modules used: fgsl galacticus_display
iso_c_binding node_component_disk_standard_data

subroutine: node_component_disk_standard_thread_initialize
Description: Initializes the standard disk component module for each thread.
Code lines: 59
Contained by: module node_component_disk_standard
Modules used: galacticus_error galacticus_nodes
input_parameters node_component_disk_standard_data

subroutine: node_component_disk_standard_thread_uninitialize
Description: Uninitializes the standard disk component module for each thread.
Code lines: 19
Contained by: module node_component_disk_standard

Modules used: galacticus_nodes node_component_disk_standard_data

function: node_component_disk_standard_velocity
Description: Return the circular velocity of the standard disk.
Code lines: 10
Contained by: module node_component_disk_standard
Modules used: galacticus_nodes

subroutine: node_component_disk_standard_velocity_set
Description: Set the circular velocity of the standard disk.
Code lines: 11
Contained by: module node_component_disk_standard
Modules used: galacticus_nodes

file: objects.nodes.components.disk.standard.data.F90
Description: Contains a module which stores data for the standard disk node component.
Code lines: 61

module: node_component_disk_standard_data
Description: Stores data for the standard disk node component.
Code lines: 39
Contained by: file objects.nodes.components.disk.standard.data.F90
Modules used: kind_numbers mass_distributions
Used by: function node_component_disk_standard_-density; function node_component_disk_standard_-enclosed_mass; function node_component_disk_standard_-half_mass_radius; function node_component_disk_standard_-potential; function node_component_disk_standard_-rotation_curve; function node_component_disk_standard_-rotation_curve_gradient; function node_component_disk_standard_-surface_density; subroutine node_component_disk_-standard_calculation_reset; subroutine node_component_disk_-standard_initialize; subroutine node_component_disk_-standard_radius_solver; subroutine node_component_disk_-standard_state_retrieve; subroutine node_component_disk_-standard_state_store; subroutine node_component_disk_-standard_thread_initialize; subroutine node_component_disk_-standard_thread_uninitialize

subroutine: node_component_disk_standard_reset
Description: Reset calculations for the standard disk component.
Code lines: 11
Contained by: module node_component_disk_standard_data

file: objects.nodes.components.disk.very_simple.F90
Description: Contains a module that implements a very simple disk component.
Code lines: 796

module: node_component_disk_very_simple

Description: Implements a very simple disk component.
Code lines: 774
Contained by: file objects.nodes.components.disk.very_simple.F90
Modules used: cosmology_functions, dark_matter_halo_scales, dark_matter_profiles_dmo, galacticus_nodes, iso_varying_string, math_exponentiation, star_formation_feedback_disks, star_formation_timescales_disks, stellar_population_properties
Used by: subroutine satellitemergerprocess, subroutine standardderivativescompute, subroutine standardevolve, subroutine standardpoststepprocessing, subroutine node_components_initialize, subroutine node_components_thread_initialize, subroutine node_components_thread_uninitialize

subroutine: node_component_disk_very_simple_analytic_solver
Code lines: 98
Contained by: module node_component_disk_very_simple
Modules used: abundances_structure, galacticus_error, galacticus_nodes, histories, stellar_luminosities_structure

subroutine: node_component_disk_very_simple_create
Description: Create properties in a very simple disk component.
Code lines: 27
Contained by: module node_component_disk_very_simple
Modules used: galacticus_nodes, histories

subroutine: node_component_disk_very_simple_initialize
Description: Initializes the tree node very simple disk component module.
Code lines: 102
Contained by: module node_component_disk_very_simple
Modules used: galacticus_nodes, input_parameters

subroutine: node_component_disk_very_simple_post_evolve
Description: Catch rounding errors in the very simple disk gas evolution.
Code lines: 26
Contained by: module node_component_disk_very_simple
Modules used: abundances_structure, galacticus_display, galacticus_nodes, histories, stellar_luminosities_structure, string_handling

subroutine: node_component_disk_very_simple_post_step
Description: Catch rounding errors in the very simple disk gas evolution.
Code lines: 64
Contained by: module node_component_disk_very_simple
Modules used: abundances_structure, fgsl, galacticus_display, galacticus_nodes

stellar_luminosities_structure string_handling

subroutine: node_component_disk_very_simple_pre_evolve
Description: Ensure the disk has been initialized.
Code lines: 16
Contained by: module node_component_disk_very_simple
Modules used: galacticus_nodes

subroutine: node_component_disk_very_simple_rate_compute
Description: Compute the very simple disk node mass rate of change.
Code lines: 70
Contained by: module node_component_disk_very_simple
Modules used: abundances_structure dark_matter_halo_scales galactic_structure_options galacticus_nodes histories star_formation_feedback_disks stellar_feedback stellar_luminosities_structure

subroutine: node_component_disk_very_simple_rates
Description: Compute rates.
Code lines: 49
Contained by: module node_component_disk_very_simple
Modules used: abundances_structure galactic_structure_options galacticus_nodes histories stellar_feedback stellar_luminosities_structure

subroutine: node_component_disk_very_simple_satellite_merging
Description: Transfer any very simple disk associated with node to its host halo.
Code lines: 53
Contained by: module node_component_disk_very_simple
Modules used: abundances_structure galacticus_error galacticus_nodes satellite_merging_mass_movements satellite_merging_remnant_properties stellar_luminosities_structure

subroutine: node_component_disk_very_simple_scale_set
Description: Set scales for properties of node.
Code lines: 39
Contained by: module node_component_disk_very_simple
Modules used: abundances_structure galacticus_nodes histories stellar_luminosities_structure

function: node_component_disk_very_simple_sfr
Description: Return the star formation rate of the very simple disk.
Code lines: 35
Contained by: module node_component_disk_very_simple
Modules used: galacticus_nodes numerical_constants_math

subroutine: node_component_disk_very_simple_thread_initialize
Description: Initializes the tree node very simple disk profile module.
Code lines: 20

Contained by: module node_component_disk_very_simple
Modules used: galacticus_nodes input_parameters

subroutine: node_component_disk_very_simple_thread_uninitialize
Description: Uninitializes the tree node very simple disk profile module.
Code lines: 14
Contained by: module node_component_disk_very_simple
Modules used: galacticus_nodes

file: objects.nodes.components.disk.very_simple.size.F90
Description: Contains a module that implements a very simple disk component.
Code lines: 235

module: node_component_disk_very_simple_size
Description: Implements a very simple disk component.
Code lines: 213
Contained by: file objects.nodes.components.disk.very_simple.size.F90
Modules used: dark_matter_profiles_dmo iso_varying_string
Used by: subroutine equilibriumsolve subroutine fixedsolve subroutine linearsolve subroutine simplesolve subroutine node_components_initialize subroutine node_components_thread_initialize subroutine node_components_thread_uninitialize

subroutine: node_component_disk_very_simple_size_initialize
Description: Initializes the tree node exponential disk methods module.
Code lines: 19
Contained by: module node_component_disk_very_simple_size
Modules used: galacticus_nodes input_parameters

function: node_component_disk_very_simple_size_radius
Description: Return the radius of the disk used in structure solvers.
Code lines: 10
Contained by: module node_component_disk_very_simple_size
Modules used: galacticus_nodes

subroutine: node_component_disk_very_simple_size_radius_set
Description: Set the radius of the disk used in structure solvers.
Code lines: 11
Contained by: module node_component_disk_very_simple_size
Modules used: galacticus_nodes

subroutine: node_component_disk_very_simple_size_radius_solver
Description: Interface for the size solver algorithm.
Code lines: 31
Contained by: module node_component_disk_very_simple_size
Modules used: dark_matter_halo_spins galacticus_nodes

subroutine: `node_component_disk_very_simple_size_radius_solver_plausibility`
Description: Determines whether the disk is physically plausible for radius solving tasks. Require that it have non-zero mass.
Code lines: 17
Contained by: module `node_component_disk_very_simple_size`
Modules used: `galacticus_nodes`

subroutine: `node_component_disk_very_simple_size_thread_initialize`
Description: Initializes the tree node standard merging statistics module.
Code lines: 11
Contained by: module `node_component_disk_very_simple_size`
Modules used: `galacticus_nodes` `input_parameters`

subroutine: `node_component_disk_very_simple_size_thread_uninitialize`
Description: Uninitializes the tree node standard merging statistics module.
Code lines: 9
Contained by: module `node_component_disk_very_simple_size`
Modules used: `galacticus_nodes`

function: `node_component_disk_very_simple_size_velocity`
Description: Return the circular velocity of the disk.
Code lines: 10
Contained by: module `node_component_disk_very_simple_size`
Modules used: `galacticus_nodes`

subroutine: `node_component_disk_very_simple_size_velocity_set`
Description: Set the circular velocity of the disk.
Code lines: 11
Contained by: module `node_component_disk_very_simple_size`
Modules used: `galacticus_nodes`

file: `objects.nodes.components.dynamics_statistics.bars.F90`
Description: Contains a module which implements tracking of dynamics statistics related to bars.
Code lines: 280

module: `node_component_dynamics_statistics_bars`
Description: Implements tracking of dynamics statistics related to bars.
Code lines: 258
Contained by: file `objects.nodes.components.dynamics_statistics.bars.F90`
Modules used: `dark_matter_halo_scales` `galactic_dynamics_bar_instabilities`
Used by: subroutine `standardderivativescompute` subroutine `node_components_initialize` subroutine `node_components_thread_initialize` subroutine `node_components_thread_uninitialize` subroutine `evolveforestsperform`

subroutine: `node_component_dynamics_statistics_bars_initialize`
Description: Initializes the tree node standard disk methods module.
Code lines: 19

Contained by: module node_component_dynamics_statistics_bars
Modules used: galacticus_nodes input_parameters

subroutine: node_component_dynamics_statistics_bars_output
Description: Store the dynamical histories of galaxies to GALACTICUS output file.
Code lines: 61
Contained by: module node_component_dynamics_statistics_bars
Modules used: galacticus_hdf5 galacticus_nodes
io_hdf5 iso_c_binding
iso_varying_string numerical_constants_astronomical
string_handling

subroutine: node_component_dynamics_statistics_bars_rate_compute
Description: Compute the standard disk node mass rate of change.
Code lines: 42
Contained by: module node_component_dynamics_statistics_bars
Modules used: galacticus_error galacticus_nodes

subroutine: node_component_dynamics_statistics_bars_record
Description: Record the bar dynamical state of a satellite galaxy.
Code lines: 37
Contained by: module node_component_dynamics_statistics_bars
Modules used: galacticus_nodes kepler_orbits
numerical_constants_math numerical_interpolation
satellite_orbits

subroutine: node_component_dynamics_statistics_bars_thread_initialize
Description: Initializes the tree node very simple disk profile module.
Code lines: 12
Contained by: module node_component_dynamics_statistics_bars
Modules used: galacticus_nodes input_parameters

subroutine: node_component_dynamics_statistics_bars_thread_uninitialize
Description: Uninitializes the tree node very simple disk profile module.
Code lines: 10
Contained by: module node_component_dynamics_statistics_bars
Modules used: galacticus_nodes

file: objects.nodes.components.formation_times.Cole2000.F90
Description: Contains a module of halo formation time methods.
Code lines: 188

module: node_component_formation_times_cole2000
Description: Implement tracking of halo formation times.
Code lines: 166
Contained by: file objects.nodes.components.formation_times.Cole2000.F90
Used by: subroutine merger_tree_initialize subroutine standardderivativescompute
subroutine standardpromote subroutine node_components_initialize

subroutine: `node_component_formation_time_cole2000_create`
Description: Creates a halo formation time component for `node`. This function is also used to "reform" the halo, since it simply resets the formation time and mass to the current values.
Code lines: 28
Contained by: module node_component_formation_times_cole2000
Modules used: events_halo_formation galacticus_nodes

subroutine: `node_component_formation_time_cole2000_node_promotion`
Code lines: 19
Contained by: module node_component_formation_times_cole2000
Modules used: galacticus_nodes

subroutine: `node_component_formation_time_cole2000_rate_compute`
Description: Check for need to update the formation time of a node in the `Cole2000` formation time component.
Code lines: 32
Contained by: module node_component_formation_times_cole2000
Modules used: galacticus_nodes

subroutine: `node_component_formation_time_cole2000_tree_initialize`
Description: Initialize the formation node pointer for any childless node.
Code lines: 10
Contained by: module node_component_formation_times_cole2000
Modules used: galacticus_nodes

subroutine: `node_component_formation_times_cole2000_initialize`
Description: Initializes the tree node formation time tracking module.
Code lines: 24
Contained by: module node_component_formation_times_cole2000
Modules used: input_parameters

file: `objects.nodes.components.formation_times.mass_fraction.F90`
Description: Contains a module of halo formation time methods.
Code lines: 131

module: `node_component_formation_times_mass_fraction`
Description: Implement tracking of halo formation times.
Code lines: 110
Contained by: file objects.nodes.components.formation_times.mass_fraction.F90
Modules used: galacticus_nodes
Used by: subroutine merger_tree_initialize subroutine standardpromote subroutine node_components_initialize

subroutine: `node_component_formation_time_mass_fraction_node_promotion`
Description: Handle node promotion for formation times.
Code lines: 22
Contained by: module node_component_formation_times_mass_fraction
Modules used: galacticus_error iso_varying_string

string_handling

subroutine: node_component_formation_time_mass_fraction_tree_initialize
Description: Initialize the formation node pointer for any childless node.
Code lines: 31
Contained by: module node_component_formation_times_mass_fraction

subroutine: node_component_formation_times_mass_fraction_initialize
Description: Initializes the tree node formation time tracking module.
Code lines: 15
Contained by: module node_component_formation_times_mass_fraction
Modules used: input_parameters

file: objects.nodes.components.host_history.standard.F90
Description: Contains a module which implements a component class that tracks the maximum host mass seen by each halo.
Code lines: 101

module: node_component_host_history_standard
Description: Implements a component class that tracks the maximum host mass seen by each halo.
Code lines: 79
Contained by: file objects.nodes.components.host_history.standard.F90
Used by: subroutine merger_tree_initialize; subroutine standardevolve; subroutine standardmerge; subroutine standardpromote

subroutine: node_component_host_history_standard_merger_tree_init
Description: Initialize the standard host history component by creating components in nodes and assigning host mass for satellites.
Code lines: 22
Contained by: module node_component_host_history_standard
Modules used: galacticus_nodes

subroutine: node_component_host_history_standard_update_history
Description: Record any major merger of thisNode.
Code lines: 17
Contained by: module node_component_host_history_standard
Modules used: galacticus_nodes

file: objects.nodes.components.hot_halo.cold_mode.F90
Description: Contains a module which implements an extension to the standard hot halo node component which supports a cold mode reservoir.
Code lines: 645

module: node_component_hot_halo_cold_mode
Description: Implements an extension to the standard hot halo node component which supports a cold mode reservoir.
Code lines: 622
Contained by: file objects.nodes.components.hot_halo.cold_mode.F90
Modules used: accretion_halos; cooling_cold_mode_infall_rates; cosmology_parameters; dark_matter_halo_scales

	dark_matter_profiles_dmo	iso_varying_string
	radiation_fields	
Used by:	subroutine event_halo_formation	subroutine satellitemergerprocess
	subroutine merger_tree_initialize	subroutine standardderivativescompute
	subroutine standardevolve	subroutine standardmerge
	subroutine standardpromote	subroutine node_components_initialize
	subroutine node_components_thread_-initialize	subroutine node_components_thread_-uninitialize

subroutine: node_component_hot_halo_cold_mode_formation

Description: Updates the hot halo gas distribution at a formation event, if requested.
Code lines: 27
Contained by: module node_component_hot_halo_cold_mode
Modules used:

abundances_structure	dark_matter_halo_scales
galacticus_nodes	node_component_hot_halo_standard_data
numerical_constants_astronomical	

subroutine: node_component_hot_halo_cold_mode_initialize

Description: Initializes the tree node hot halo methods module.
Code lines: 29
Contained by: module node_component_hot_halo_cold_mode
Modules used:

abundances_structure	galacticus_nodes
input_parameters	

subroutine: node_component_hot_halo_cold_mode_node_merger

Description: Starve node by transferring its hot halo to its parent.
Code lines: 80
Contained by: module node_component_hot_halo_cold_mode
Modules used:

abundances_structure	dark_matter_halo_spins
galactic_structure_enclosed_masses	galactic_structure_options
galacticus_nodes	node_component_hot_halo_standard_data

subroutine: node_component_hot_halo_cold_mode_outflow_return

Description: Return outflowed gas to the cold mode reservoir.
Code lines: 64
Contained by: module node_component_hot_halo_cold_mode
Modules used:

abundances_structure	galactic_structure_densities
galactic_structure_options	galacticus_error
galacticus_nodes	node_component_hot_halo_standard_data
numerical_constants_astronomical	numerical_constants_atomic
numerical_constants_math	numerical_constants_prefixes

subroutine: node_component_hot_halo_cold_mode_promote

Description: Ensure that node is ready for promotion to its parent. In this case, we simply update the cold mode mass of node to account for any cold mode gas already in the parent.
Code lines: 29
Contained by: module node_component_hot_halo_cold_mode
Modules used:

dark_matter_halo_scales	galacticus_nodes

subroutine: `node_component_hot_halo_cold_mode_push_to_cooling_pipes`

Description: Push mass through the cooling pipes (along with appropriate amounts of metals and angular momentum) at the given rate.
Code lines: 52
Contained by: module node_component_hot_halo_cold_mode
Modules used: abundances_structure, galacticus_error, galacticus_nodes, node_component_hot_halo_standard_data

subroutine: `node_component_hot_halo_cold_mode_rate_compute`

Description: Compute the hot halo node mass rate of change.
Code lines: 75
Contained by: module node_component_hot_halo_cold_mode
Modules used: abundances_structure, accretion_halos, dark_matter_halo_spins, galactic_structure_densities, galactic_structure_options, galacticus_nodes, hot_halo_ram_pressure_stripping, node_component_hot_halo_standard_data, numerical_constants_astronomical

subroutine: `node_component_hot_halo_cold_mode_satellite_merging`

Description: Remove any cold mode gas associated with `node` before it merges with its host halo.
Code lines: 31
Contained by: module node_component_hot_halo_cold_mode
Modules used: abundances_structure, galacticus_nodes, node_component_hot_halo_standard_data

subroutine: `node_component_hot_halo_cold_mode_scale_set`

Description: Set scales for properties of `node`.
Code lines: 28
Contained by: module node_component_hot_halo_cold_mode
Modules used: abundances_structure, galacticus_nodes

subroutine: `node_component_hot_halo_cold_mode_thread_initialize`

Description: Initializes the tree node hot halo cold mode methods module.
Code lines: 18
Contained by: module node_component_hot_halo_cold_mode
Modules used: galacticus_nodes, hot_halo_cold_mode_density_core_radii, input_parameters, node_component_hot_halo_cold_mode_-structure_tasks

subroutine: `node_component_hot_halo_cold_mode_thread_uninitialize`

Description: Uninitializes the tree node hot halo cold mode methods module.
Code lines: 15
Contained by: module node_component_hot_halo_cold_mode
Modules used: galacticus_nodes, node_component_hot_halo_cold_mode_-structure_tasks

subroutine: `node_component_hot_halo_cold_mode_tree_initialize`

Description: Initialize the contents of the hot halo component for any sub-resolution accretion (i.e. the gas that would have been accreted if the merger tree had infinite resolution).
Code lines: 45
Contained by: module node_component_hot_halo_cold_mode
Modules used: abundances_structure, dark_matter_halo_spins, galacticus_nodes

file: objects.nodes.components.hot_halo.cold_mode.structure_tasks.F90
Description: Contains a module which implements structure tasks for the cold mode hot halo component.
Code lines: 167

module: node_component_hot_halo_cold_mode_structure_tasks
Description: Implements structure tasks for the cold mode hot halo component.
Code lines: 145
Contained by: file objects.nodes.components.hot_halo.cold_mode.structure_tasks.F90
Modules used: hot_halo_cold_mode_density_core_radii, mass_distributions
Used by: subroutine adiabaticgnedin2004computefactors; function galactic_structure_density; function galactic_structure_enclosed_mass; function galactic_structure_rotation_curve; function galactic_structure_rotation_curve_gradient; subroutine node_component_hot_halo_cold_mode_thread_initialize; subroutine node_component_hot_halo_cold_mode_thread_uninitialize

function: node_component_hot_halo_cold_mode_density_task
Description: Computes the density at a given position for a dark matter profile.
Code lines: 33
Contained by: module node_component_hot_halo_cold_mode_structure_tasks
Modules used: coordinates, galactic_structure_options, galacticus_nodes

function: node_component_hot_halo_cold_mode_enclosed_mass_task
Description: Computes the mass within a given radius for the cold mode hot halo component.
Code lines: 36
Contained by: module node_component_hot_halo_cold_mode_structure_tasks
Modules used: galactic_structure_options, galacticus_nodes

function: node_component_hot_halo_cold_mode_rotation_curve_gradient_task
Description: Computes the rotation curve gradient at a given radius for the hot halo density profile.
Code lines: 23
Contained by: module node_component_hot_halo_cold_mode_structure_tasks
Modules used: galactic_structure_options, galacticus_nodes, numerical_constants_math, numerical_constants_physical

function: node_component_hot_halo_cold_mode_rotation_curve_task
Description: Computes the rotation curve at a given radius for the hot halo density profile.
Code lines: 19
Contained by: module node_component_hot_halo_cold_mode_structure_tasks

Modules used: galactic_structure_options galacticus_nodes numerical_constants_physical

file: objects.nodes.components.hot_halo.outflow_tracking.F90
Description: Contains a module which implements an extension of the standard hot halo node component which tracks the metals arriving from outflows.
Code lines: 167

module: node_component_hot_halo_outflow_tracking
Description: Implements an extension of the standard hot halo node component which tracks the metals arriving from outflows.
Code lines: 144
Contained by: file objects.nodes.components.hot_halo.outflow_tracking.F90
Modules used: dark_matter_halo_scales
Used by: subroutine standardderivativescompute subroutine standardevolve subroutine node_components_thread_initialize subroutine node_components_thread_uninitialize

subroutine: node_component_hot_halo_outflow_tracking_rate_compute
Description: Compute the hot halo node mass rate of change.
Code lines: 33
Contained by: module node_component_hot_halo_outflow_tracking
Modules used: abundances_structure galacticus_nodes node_component_hot_halo_standard_data

subroutine: node_component_hot_halo_outflow_tracking_scale_set
Description: Set scales for properties of node.
Code lines: 27
Contained by: module node_component_hot_halo_outflow_tracking
Modules used: abundances_structure chemical_abundances_structure dark_matter_halo_scales galacticus_nodes

subroutine: node_component_hot_halo_outflow_tracking_thread_initialize
Description: Initializes the tree node very simple disk profile module.
Code lines: 11
Contained by: module node_component_hot_halo_outflow_tracking
Modules used: galacticus_nodes input_parameters

subroutine: node_component_hot_halo_outflow_tracking_thread_uninitialize
Description: Uninitializes the tree node very simple disk profile module.
Code lines: 9
Contained by: module node_component_hot_halo_outflow_tracking
Modules used: galacticus_nodes

file: objects.nodes.components.hot_halo.standard.F90
Description: Contains a module which implements the standard hot halo node component.
Code lines: 1687

module: node_component_hot_halo_standard

Description: Implements the standard hot halo node component.
Code lines: 1665
Contained by: file objects.nodes.components.hot_halo.standard.F90
Modules used: accretion_halos, chemical_reaction_rates, chemical_states, cooling_infall_radii, cooling_rates, cooling_specific_angular_momenta, cosmology_functions, cosmology_parameters, dark_matter_halo_scales, dark_matter_profiles_dmo, hot_halo_mass_distributions, hot_halo_outflows_reincorporations, hot_halo_ram_pressure_stripping, hot_halo_ram_pressure_stripping_timescales, kind_numbers, radiation_fields
Used by: subroutine event_halo_formation, subroutine galacticus_calculations_reset, subroutine satellitemergerprocess, subroutine merger_tree_initialize, subroutine standardderivativescompute, subroutine standardevolve, subroutine standardmerge, subroutine standardpoststepprocessing, subroutine standardpromote, subroutine node_components_initialize, subroutine node_components_thread_initialize, subroutine node_components_thread_uninitialize

subroutine: node_component_hot_halo_standard_cooling_rate
Description: Get and store the cooling rate for node.
Code lines: 25
Contained by: module node_component_hot_halo_standard
Modules used: galacticus_nodes

subroutine: node_component_hot_halo_standard_create
Description: Creates a hot halo component for node.
Code lines: 15
Contained by: module node_component_hot_halo_standard
Modules used: galacticus_nodes

subroutine: node_component_hot_halo_standard_formation
Description: Updates the hot halo gas distribution at a formation event, if requested.
Code lines: 64
Contained by: module node_component_hot_halo_standard
Modules used: abundances_structure, chemical_abundances_structure, chemical_reaction_rates_utilities, galacticus_nodes, node_component_hot_halo_standard_data, numerical_constants_astronomical

subroutine: node_component_hot_halo_standard_heat_source
Description: An incoming pipe for sources of heating to the hot halo.
Code lines: 49
Contained by: module node_component_hot_halo_standard
Modules used: galacticus_error, galacticus_nodes

subroutine: node_component_hot_halo_standard_hot_gas_all_rate

Description: Adjusts the rates of all components of the hot gas reservoir under the assumption of uniformly distributed properties (e.g. fully-mixed metals).
Code lines: 27
Contained by: module node_component_hot_halo_standard
Modules used: galacticus_nodes

subroutine: node_component_hot_halo_standard_initialize
Description: Initializes the standard hot halo component module.
Code lines: 163
Contained by: module node_component_hot_halo_standard
Modules used: abundances_structure, chemical_abundances_structure, galacticus_error, galacticus_nodes, input_parameters, iso_varying_string, node_component_hot_halo_standard_data

subroutine: node_component_hot_halo_standard_initializor
Description: Initializes a standard hot halo component.
Code lines: 16
Contained by: module node_component_hot_halo_standard
Modules used: galacticus_nodes

subroutine: node_component_hot_halo_standard_mass_sink
Description: Account for a sink of gaseous material in the standard hot halo hot gas.
Code lines: 18
Contained by: module node_component_hot_halo_standard
Modules used: galacticus_error, galacticus_nodes

subroutine: node_component_hot_halo_standard_node_merger
Description: Starve node by transferring its hot halo to its parent.
Code lines: 115
Contained by: module node_component_hot_halo_standard
Modules used: abundances_structure, chemical_abundances_structure, dark_matter_halo_spins, galactic_structure_enclosed_masses, galactic_structure_options, galacticus_nodes, node_component_hot_halo_standard_data

function: node_component_hot_halo_standard_outer_radius
Description: Return the outer radius in the standard hot halo.
Code lines: 12
Contained by: module node_component_hot_halo_standard
Modules used: galacticus_nodes

function: node_component_hot_halo_standard_outer_radius_growth_rate
Description: Compute the growth rate of the outer radius of the hot halo.
Code lines: 27
Contained by: module node_component_hot_halo_standard
Modules used: galacticus_nodes

subroutine: `node_component_hot_halo_standard_outflow_return`
Description: Return outflowed gas to the hot halo.
Code lines: 90
Contained by: module `node_component_hot_halo_standard`
Modules used: `abundances_structure` `chemical_abundances_structure` `chemical_reaction_rates_utilities` `galacticus_error` `galacticus_nodes` `node_component_hot_halo_standard_data` `numerical_constants_astronomical` `numerical_constants_atomic` `numerical_constants_prefixes`

function: `node_component_hot_halo_standard_outflow_stripped_fraction`
Description: Compute the fraction of material outflowing into the hot halo of `node` which is susceptible to being stripped away.
Code lines: 19
Contained by: module `node_component_hot_halo_standard`
Modules used: `galacticus_nodes`

subroutine: `node_component_hot_halo_standard_outflowing_abundances_rate`
Description: Accept outflowing gas abundances from a galaxy and deposit it into the outflowed reservoir.
Code lines: 26
Contained by: module `node_component_hot_halo_standard`
Modules used: `abundances_structure` `galacticus_nodes`

subroutine: `node_component_hot_halo_standard_outflowing_ang_mom_rate`
Description: Accept outflowing gas angular momentum from a galaxy and deposit it into the outflowed reservoir.
Code lines: 25
Contained by: module `node_component_hot_halo_standard`
Modules used: `galacticus_nodes` `node_component_hot_halo_standard_data`

subroutine: `node_component_hot_halo_standard_outflowing_mass_rate`
Description: Accept outflowing gas from a galaxy and deposit it into the outflowed and stripped reservoirs.
Code lines: 26
Contained by: module `node_component_hot_halo_standard`
Modules used: `galacticus_nodes`

subroutine: `node_component_hot_halo_standard_post_evolve`
Description: Do processing of the node required after evolution.
Code lines: 54
Contained by: module `node_component_hot_halo_standard`
Modules used: `abundances_structure` `galacticus_nodes` `node_component_hot_halo_standard_data`

subroutine: `node_component_hot_halo_standard_post_step`
Description: Do processing of the node required after evolution.
Code lines: 23
Contained by: module `node_component_hot_halo_standard`
Modules used: `fgsl` `galacticus_nodes`

subroutine: `node_component_hot_halo_standard_pre_evolve`
Description: Ensure the standard hot halo has been initialized.
Code lines: 16
Contained by: module `node_component_hot_halo_standard`
Modules used: `galacticus_nodes`

subroutine: `node_component_hot_halo_standard_promote`
Description: Ensure that `node` is ready for promotion to its parent. In this case, we simply update the hot halo mass of `node` to account for any hot halo already in the parent.
Code lines: 49
Contained by: module `node_component_hot_halo_standard`
Modules used: `abundances_structure` `chemical_abundances_structure` `galacticus_nodes`

subroutine: `node_component_hot_halo_standard_push_from_halo`
Description: Push mass from the hot halo into an infinite sink (along with appropriate amounts of metals, chemicals and angular momentum) at the given rate.
Code lines: 35
Contained by: module `node_component_hot_halo_standard`
Modules used: `abundances_structure` `chemical_abundances_structure` `galacticus_nodes`

subroutine: `node_component_hot_halo_standard_push_to_cooling_pipes`
Description: Push mass through the cooling pipes (along with appropriate amounts of metals and angular momentum) at the given rate.
Code lines: 67
Contained by: module `node_component_hot_halo_standard`
Modules used: `abundances_structure` `galacticus_error` `galacticus_nodes` `node_component_hot_halo_standard_data`

subroutine: `node_component_hot_halo_standard_rate_compute`
Description: Compute the hot halo node mass rate of change.
Code lines: 120
Contained by: module `node_component_hot_halo_standard`
Modules used: `abundances_structure` `chemical_abundances_structure` `chemical_reaction_rates_utilities` `dark_matter_halo_spins` `galacticus_nodes` `node_component_hot_halo_standard_data` `numerical_constants_astronomical`

subroutine: `node_component_hot_halo_standard_reset`
Description: Remove memory of stored computed values as we're about to begin computing derivatives anew.
Code lines: 11
Contained by: module `node_component_hot_halo_standard`
Modules used: `galacticus_nodes`

subroutine: `node_component_hot_halo_standard_satellite_merging`
Description: Remove any hot halo associated with `node` before it merges with its host halo.

Code lines: 44
Contained by: module node_component_hot_halo_standard
Modules used: abundances_structure, chemical_abundances_structure, galacticus_nodes, node_component_hot_halo_standard_data

subroutine: node_component_hot_halo_standard_scale_set
Description: Set scales for properties of node.
Code lines: 37
Contained by: module node_component_hot_halo_standard
Modules used: abundances_structure, chemical_abundances_structure, galacticus_nodes

subroutine: node_component_hot_halo_standard_strip_gas_rate
Description: Add gas stripped from the hot halo to the stripped gas reservoirs under the assumption of uniformly distributed properties (e.g. fully-mixed metals).
Code lines: 30
Contained by: module node_component_hot_halo_standard
Modules used: galacticus_nodes

subroutine: node_component_hot_halo_standard_thread_initialize
Description: Initializes the tree node hot halo methods module.
Code lines: 46
Contained by: module node_component_hot_halo_standard
Modules used: galacticus_error, galacticus_nodes, input_parameters

subroutine: node_component_hot_halo_standard_thread_uninitialize
Description: Uninitializes the tree node hot halo methods module.
Code lines: 25
Contained by: module node_component_hot_halo_standard
Modules used: galacticus_nodes

subroutine: node_component_hot_halo_standard_tree_initialize
Description: Initialize the contents of the hot halo component for any sub-resolution accretion (i.e. the gas that would have been accreted if the merger tree had infinite resolution).
Code lines: 56
Contained by: module node_component_hot_halo_standard
Modules used: abundances_structure, chemical_abundances_structure, dark_matter_halo_spins, galacticus_nodes

file: objects.nodes.components.hot_halo.standard.data.F90
Description: Contains a module which provides data for the standard hot halo node component.
Code lines: 42

module: node_component_hot_halo_standard_data
Description: Provides data for the standard hot halo node component.
Code lines: 20
Contained by: file objects.nodes.components.hot_halo.standard.data.F90
Used by: subroutine node_component_hot_halo_-cold_mode_formation, subroutine node_component_hot_halo_-cold_mode_node_merger

subroutine node_component_hot_halo_cold_mode_outflow_return
subroutine node_component_hot_halo_cold_mode_push_to_cooling_pipes
subroutine node_component_hot_halo_cold_mode_rate_compute
subroutine node_component_hot_halo_cold_mode_satellite_merging
subroutine node_component_hot_halo_outflow_tracking_rate_compute
subroutine node_component_hot_halo_standard_formation
subroutine node_component_hot_halo_standard_initialize
subroutine node_component_hot_halo_standard_node_merger
subroutine node_component_hot_halo_standard_outflow_return
subroutine node_component_hot_halo_standard_outflowing_ang_mom_rate
subroutine node_component_hot_halo_standard_post_evolve
subroutine node_component_hot_halo_standard_push_to_cooling_pipes
subroutine node_component_hot_halo_standard_rate_compute
subroutine node_component_hot_halo_standard_satellite_merging

file: objects.nodes.components.hot_halo.very_simple.F90
Description: Contains a module which implements a very simple hot halo node component.
Code lines: 551

module: node_component_hot_halo_very_simple
Description: Implements a very simple hot halo node component.
Code lines: 529
Contained by: file objects.nodes.components.hot_halo.very_simple.F90
Modules used: accretion_halos cooling_rates dark_matter_halo_scales
Used by: subroutine galacticus_calculations_reset; subroutine satellitemergerprocess; subroutine merger_tree_initialize; subroutine standardderivativescompute; subroutine standardevolve; subroutine standardmerge; subroutine standardpromote; subroutine node_components_thread_initialize; subroutine node_components_thread_uninitialize

subroutine: node_component_hot_halo_very_simple_cooling_rate
Description: Get and store the cooling rate for node.
Code lines: 20
Contained by: module node_component_hot_halo_very_simple
Modules used: galacticus_nodes

subroutine: node_component_hot_halo_very_simple_create
Description: Creates a very simple hot halo component for node.
Code lines: 13
Contained by: module node_component_hot_halo_very_simple
Modules used: galacticus_nodes

subroutine: node_component_hot_halo_very_simple_initialize
Description: Initializes the very simple hot halo component module.

Code lines: 12
Contained by: module node_component_hot_halo_very_simple
Modules used: galacticus_nodes

subroutine: node_component_hot_halo_very_simple_node_merger
Description: Starve node by transferring its hot halo to its parent.
Code lines: 55
Contained by: module node_component_hot_halo_very_simple
Modules used: abundances_structure galacticus_nodes

function: node_component_hot_halo_very_simple_outer_radius
Description: Return the outer radius of the hot halo. Assumes a simple model in which this always equals the virial radius.
Code lines: 8
Contained by: module node_component_hot_halo_very_simple
Modules used: galacticus_nodes

subroutine: node_component_hot_halo_very_simple_outflowing_abundances_rate
Description: Accept outflowing gas abundances from a galaxy and deposit them into very simple hot halo.
Code lines: 14
Contained by: module node_component_hot_halo_very_simple
Modules used: abundances_structure galacticus_nodes

subroutine: node_component_hot_halo_very_simple_outflowing_mass_rate
Description: Accept outflowing gas from a galaxy and deposit it into very simple hot halo.
Code lines: 13
Contained by: module node_component_hot_halo_very_simple
Modules used: galacticus_nodes

subroutine: node_component_hot_halo_very_simple_post_evolve
Description: Do processing of the node required after evolution.
Code lines: 28
Contained by: module node_component_hot_halo_very_simple
Modules used: abundances_structure galacticus_nodes

subroutine: node_component_hot_halo_very_simple_promote
Description: Ensure that node is ready for promotion to its parent. In this case, we simply update the hot halo mass of node to account for any hot halo already in the parent.
Code lines: 27
Contained by: module node_component_hot_halo_very_simple
Modules used: galacticus_nodes

subroutine: node_component_hot_halo_very_simple_push_to_cooling_pipes
Description: Push mass through the cooling pipes at the given rate.
Code lines: 28
Contained by: module node_component_hot_halo_very_simple
Modules used: abundances_structure galacticus_nodes

subroutine: `node_component_hot_halo_very_simple_rate_compute`
Description: Compute the very simple hot halo component mass rate of change.
Code lines: 32
Contained by: module `node_component_hot_halo_very_simple`
Modules used: `galacticus_nodes`

subroutine: `node_component_hot_halo_very_simple_reset`
Description: Remove memory of stored computed values as we're about to begin computing derivatives anew.
Code lines: 9
Contained by: module `node_component_hot_halo_very_simple`
Modules used: `galacticus_nodes`

subroutine: `node_component_hot_halo_very_simple_satellite_merging`
Description: Remove any hot halo associated with `node` before it merges with its host halo.
Code lines: 26
Contained by: module `node_component_hot_halo_very_simple`
Modules used: `abundances_structure` `galacticus_nodes`

subroutine: `node_component_hot_halo_very_simple_scale_set`
Description: Set scales for properties of `node`.
Code lines: 26
Contained by: module `node_component_hot_halo_very_simple`
Modules used: `abundances_structure` `galacticus_nodes`

subroutine: `node_component_hot_halo_very_simple_thread_initialize`
Description: Initializes the tree node very simple disk profile module.
Code lines: 13
Contained by: module `node_component_hot_halo_very_simple`
Modules used: `galacticus_nodes` `input_parameters`

subroutine: `node_component_hot_halo_very_simple_thread_uninitialize`
Description: Uninitializes the tree node very simple disk profile module.
Code lines: 11
Contained by: module `node_component_hot_halo_very_simple`
Modules used: `galacticus_nodes`

subroutine: `node_component_hot_halo_very_simple_tree_initialize`
Description: Initialize the contents of the very simple hot halo component.
Code lines: 48
Contained by: module `node_component_hot_halo_very_simple`
Modules used: `abundances_structure` `cosmology_parameters` `galacticus_nodes`

file: `objects.nodes.components.hot_halo.very_simple_delayed.F90`
Description: Contains a module which implements an extension to the very simple hot halo node component by including an outflowed reservoir with delayed reincorporation.
Code lines: 346

module: `node_component_hot_halo_vs_delayed`

Description: Implements an extension to the very simple hot halo node component by including an outflowed reservoir with delayed reincorporation.
Code lines: 323
Contained by: file `objects.nodes.components.hot_halo.very_simple_delayed.F90`
Used by: subroutine `satellitemergerprocess`, subroutine `merger_tree_initialize`, subroutine `standardderivativescompute`, subroutine `standardevolve`, subroutine `standardmerge`, subroutine `standardpromote`, subroutine `node_components_initialize`

subroutine: `node_component_hot_halo_vs_delayed_initialize`

Description: Initializes the very simple hot halo component module.
Code lines: 25
Contained by: module `node_component_hot_halo_vs_delayed`
Modules used: `galacticus_nodes`, `input_parameters`

subroutine: `node_component_hot_halo_vs_delayed_node_merger`

Description: Starve `node` by transferring its hot halo to its parent.
Code lines: 24
Contained by: module `node_component_hot_halo_vs_delayed`
Modules used: `abundances_structure`, `galacticus_nodes`

subroutine: `node_component_hot_halo_vs_delayed_outflowing_abundances_rate`

Description: Accept outflowing gas abundances from a galaxy and deposit them into very simple hot halo.
Code lines: 14
Contained by: module `node_component_hot_halo_vs_delayed`
Modules used: `abundances_structure`, `galacticus_nodes`

subroutine: `node_component_hot_halo_vs_delayed_outflowing_mass_rate`

Description: Accept outflowing gas from a galaxy and deposit it into very simple hot halo.
Code lines: 13
Contained by: module `node_component_hot_halo_vs_delayed`
Modules used: `galacticus_nodes`

subroutine: `node_component_hot_halo_vs_delayed_post_evolve`

Description: Do processing of the node required after evolution.
Code lines: 28
Contained by: module `node_component_hot_halo_vs_delayed`
Modules used: `abundances_structure`, `galacticus_nodes`

subroutine: `node_component_hot_halo_vs_delayed_promote`

Description: Ensure that `node` is ready for promotion to its parent. In this case, we simply update the hot halo mass of `node` to account for any hot halo already in the parent.
Code lines: 26
Contained by: module `node_component_hot_halo_vs_delayed`
Modules used: `galacticus_nodes`

subroutine: `node_component_hot_halo_vs_delayed_rate_compute`

Description: Compute the very simple hot halo component mass rate of change.
Code lines: 40
Contained by: module node_component_hot_halo_vs_delayed
Modules used: abundances_structure galacticus_nodes hot_halo_outflows_reincorporations

subroutine: node_component_hot_halo_vs_delayed_satellite_merging
Description: Remove any hot halo associated with node before it merges with its host halo.
Code lines: 24
Contained by: module node_component_hot_halo_vs_delayed
Modules used: abundances_structure galacticus_nodes

subroutine: node_component_hot_halo_vs_delayed_scale_set
Description: Set scales for properties of node.
Code lines: 24
Contained by: module node_component_hot_halo_vs_delayed
Modules used: abundances_structure galacticus_nodes

subroutine: node_component_hot_halo_vs_delayed_tree_initialize
Description: Initialize the contents of the very simple hot halo component.
Code lines: 21
Contained by: module node_component_hot_halo_vs_delayed
Modules used: abundances_structure cosmology_parameters galacticus_nodes

file: objects.nodes.components.indices.standard.F90
Description: Contains a module which implements the standard indices component.
Code lines: 70

module: node_component_indices_standard
Description: Implements the standard indices component.
Code lines: 48
Contained by: file objects.nodes.components.indices.standard.F90
Used by: subroutine merger_tree_initialize

subroutine: node_component_indices_standard_merger_tree_init
Description: Initialize the indices component by creating components in nodes and storing indices.
Code lines: 20
Contained by: module node_component_indices_standard
Modules used: galacticus_nodes

file: objects.nodes.components.interoutput.standard.F90
Description: Contains a module which implements the standard indices component.
Code lines: 243

module: node_component_inter_output_standard
Description: Implements the standard indices component.
Code lines: 221
Contained by: file objects.nodes.components.interoutput.standard.F90

Modules used: output_times
Used by: subroutine satellitemergerprocess; subroutine standardderivativescompute; subroutine standardevolve; subroutine node_components_thread_initialize; subroutine node_components_thread_uninitialize; subroutine evolveforestsperform

subroutine: node_component_inter_output_standard_rate_compute
Description: Compute the exponential disk node mass rate of change.
Code lines: 40
Contained by: module node_component_inter_output_standard
Modules used: galacticus_nodes

subroutine: node_component_inter_output_standard_reset
Description: Reset interoutput accumulated quantities.
Code lines: 21
Contained by: module node_component_inter_output_standard
Modules used: galacticus_nodes; iso_c_binding; kind_numbers

subroutine: node_component_inter_output_standard_satellite_merging
Description: Remove any inter-output quantities associated with node and add them to the merge target.
Code lines: 46
Contained by: module node_component_inter_output_standard
Modules used: galacticus_error; galacticus_nodes; satellite_merging_mass_movements; satellite_merging_remnant_properties

subroutine: node_component_inter_output_standard_scale_set
Description: Set scales for properties of node.
Code lines: 26
Contained by: module node_component_inter_output_standard
Modules used: galacticus_nodes

subroutine: node_component_interoutput_standard_thread_initialize
Description: Initializes the tree node standard interoutput module.
Code lines: 11
Contained by: module node_component_inter_output_standard
Modules used: galacticus_nodes; input_parameters

subroutine: node_component_interoutput_standard_thread_uninitialize
Description: Uninitializes the tree node standard interoutput module.
Code lines: 9
Contained by: module node_component_inter_output_standard
Modules used: galacticus_nodes

file: objects.nodes.components.mass_flow_statistics.standard.F90
Description: Contains a module which implements the standard mass flow statistics component.
Code lines: 208

module: node_component_mass_flow_statistics_standard

Description: Implements the standard mass flow statistics component.
Code lines: 186
Contained by: file objects.nodes.components.mass_flow_statistics.standard.F90
Modules used: cooling_rates
Used by: subroutine merger_tree_initialize; subroutine standardderivativescompute; subroutine standardevolve; subroutine node_components_initialize; subroutine node_components_thread_initialize; subroutine node_components_thread_uninitialize; subroutine evolveforestsperform

subroutine: node_component_mass_flow_statistics_standard_extra_output

Description: Reset mass flow statistics at output time.
Code lines: 24
Contained by: module node_component_mass_flow_statistics_standard
Modules used: galacticus_nodes; iso_c_binding; kind_numbers

subroutine: node_component_mass_flow_statistics_standard_initialize

Description: Initializes the standard mass flow statistics component.
Code lines: 15
Contained by: module node_component_mass_flow_statistics_standard
Modules used: input_parameters

subroutine: node_component_mass_flow_statistics_standard_merger_tree_init

Description: Initialize the mass flow statistics component by creating components in nodes and computing formation times.
Code lines: 17
Contained by: module node_component_mass_flow_statistics_standard
Modules used: dark_matter_halo_mass_accretion_histories; galacticus_nodes

subroutine: node_component_mass_flow_statistics_standard_rate_compute

Description: Compute rates of change of properties in the standard implementation of the basic component.
Code lines: 25
Contained by: module node_component_mass_flow_statistics_standard
Modules used: galacticus_nodes

subroutine: node_component_mass_flow_statistics_standard_scale_set

Description: Set scales for properties in the standard implementation of the massFlowStatistics component.
Code lines: 19
Contained by: module node_component_mass_flow_statistics_standard
Modules used: galacticus_nodes

subroutine: node_component_mass_flow_statistics_standard_thread_initialize

Description: Initializes the tree node standard mass flow statistics module.

Code lines: 11
Contained by: module node_component_mass_flow_statistics_standard
Modules used: galacticus_nodes input_parameters

subroutine: node_component_mass_flow_statistics_standard_thread_uninit
Description: Uninitializes the tree node standard mass flow statistics module.
Code lines: 9
Contained by: module node_component_mass_flow_statistics_standard
Modules used: galacticus_nodes

file: objects.nodes.components.merging_statistics.major.F90
Description: Contains a module which implements the major merging statistics component.
Code lines: 158

module: node_component_merging_statistics_major
Description: Implements the major merging statistics component.
Code lines: 136
Contained by: file objects.nodes.components.merging_statistics.major.F90
Modules used: iso_c_binding
Used by: subroutine satellitemergerprocess subroutine standardpromote
subroutine evolveforestsperform

subroutine: node_component_merging_statistics_major_node_promotion
Description: Ensure that node is ready for promotion to its parent. In this case, we simply update the node merger time.
Code lines: 20
Contained by: module node_component_merging_statistics_major
Modules used: galacticus_nodes

subroutine: node_component_merging_statistics_major_output
Description: Output properties for all black holes in node.
Code lines: 42
Contained by: module node_component_merging_statistics_major
Modules used: galacticus_hdf5 galacticus_nodes
iso_c_binding iso_varying_string
kind_numbers string_handling

subroutine: node_component_merging_statistics_major_satellite_merging
Description: Record any major merger of node.
Code lines: 35
Contained by: module node_component_merging_statistics_major
Modules used: galacticus_nodes satellite_merging_mass_movements
satellite_merging_remnant_properties

file: objects.nodes.components.merging_statistics.recent.F90
Description: Contains a module which implements the recent merging statistics component.
Code lines: 359

module: node_component_merging_statistics_recent

Description: Implements the recent merging statistics component.
Code lines: 337
Contained by: file objects.nodes.components.merging_statistics.recent.F90
Modules used: dark_matter_halo_scales, iso_c_binding, node_component_merging_statistics_recent_data, output_times
Used by: subroutine merger_tree_initialize, subroutine standardmerge, subroutine standardpromote, subroutine standardoutput, subroutine standardpropertiescount, subroutine standardpropertynamesestablish, subroutine node_components_initialize, subroutine node_components_thread_initialize, subroutine node_components_thread_uninitialize

subroutine: node_component_merging_statistics_recent_initialize
Description: Initializes the recent merging statistics component.
Code lines: 52
Contained by: module node_component_merging_statistics_recent
Modules used: galacticus_error, input_parameters, iso_varying_string, memory_management

function: node_component_merging_statistics_recent_matches
Description: Return true if the black hole component of node is a match to the standard implementation.
Code lines: 18
Contained by: module node_component_merging_statistics_recent
Modules used: galacticus_nodes

subroutine: node_component_merging_statistics_recent_merger_tree_init
Description: Initialize the merging statistics component by creating components in nodes.
Code lines: 16
Contained by: module node_component_merging_statistics_recent
Modules used: galacticus_nodes

subroutine: node_component_merging_statistics_recent_node_merger
Description: Record any major merger of node.
Code lines: 59
Contained by: module node_component_merging_statistics_recent
Modules used: galacticus_error, galacticus_nodes, iso_c_binding

subroutine: node_component_merging_statistics_recent_node_promotion
Description: Ensure that node is ready for promotion to its parent. In this case, we simply update the node merger time.
Code lines: 16
Contained by: module node_component_merging_statistics_recent
Modules used: galacticus_nodes

subroutine: node_component_merging_statistics_recent_output

Description: Store black hole properties in the Galacticus output file buffers.
Code lines: 24
Contained by: module node_component_merging_statistics_recent
Modules used: galacticus_nodes kind_numbers multi_counters

subroutine: node_component_merging_statistics_recent_output_count
Description: Account for the number of black hole properties to be written to the the Galacticus output file.
Code lines: 11
Contained by: module node_component_merging_statistics_recent
Modules used: galacticus_nodes

subroutine: node_component_merging_statistics_recent_output_names
Description: Set names of black hole properties to be written to the Galacticus output file.
Code lines: 18
Contained by: module node_component_merging_statistics_recent
Modules used: galacticus_nodes

subroutine: node_component_merging_statistics_recent_thread_initialize
Description: Initializes the tree node recent merging flow statistics module.
Code lines: 20
Contained by: module node_component_merging_statistics_recent
Modules used: galacticus_nodes input_parameters memory_management

subroutine: node_component_merging_statistics_recent_thread_uninitialize
Description: Uninitializes the tree node recent merging flow statistics module.
Code lines: 10
Contained by: module node_component_merging_statistics_recent
Modules used: galacticus_nodes

file: objects.nodes.components.merging_statistics.recent.data.F90
Description: Contains a module which stores data for the recent merging statistics component.
Code lines: 42

module: node_component_merging_statistics_recent_data
Description: Stores data for the recent merging statistics component.
Code lines: 20
Contained by: file objects.nodes.components.merging_statistics.recent.data.F90
Modules used: iso_c_binding
Used by: function mergingstatisticsrecentmajormergercount subroutine nodecomponentmergingstatisticsrecentinitialize module node_component_merging_-statistics_recent

function: node_component_merging_statistics_recent_count
Description: Return the size of the merging statistics property
Code lines: 7

Contained by: module node_component_merging_statistics_recent_data

file: objects.nodes.components.merging_statistics.standard.F90
Description: Contains a module which implements the standard merging statistics component.
Code lines: 381

module: node_component_merging_statistics_standard
Description: Implements the standard merging statistics component.
Code lines: 359
Contained by: file objects.nodes.components.merging_statistics.standard.F90
Modules used: dark_matter_halo_mass_accretion_histories, galacticus_nodes
Used by: subroutine satellitemergerprocess, subroutine merger_tree_initialize, subroutine standardevolve, subroutine standardmerge, subroutine standardpromote, subroutine node_components_initialize, subroutine node_components_thread_initialize, subroutine node_components_thread_uninitialize

function: node_component_merging_statistics_standard_hierarchy_level
Description: Return the hierarchy level of a node, computing it if necessary.
Code lines: 19
Contained by: module node_component_merging_statistics_standard
Modules used: galacticus_nodes

function: node_component_merging_statistics_standard_hlm
Description: Return the maximum hierarchy level of a node, computing it if necessary.
Code lines: 57
Contained by: module node_component_merging_statistics_standard
Modules used: galacticus_nodes

subroutine: node_component_merging_statistics_standard_initialize
Description: Initializes the standard merging statistics component.
Code lines: 34
Contained by: module node_component_merging_statistics_standard
Modules used: input_parameters

subroutine: node_component_merging_statistics_standard_merger_tree_init
Description: Initialize the merging statistics component by creating components in nodes and computing formation times.
Code lines: 30
Contained by: module node_component_merging_statistics_standard
Modules used: dark_matter_halo_formation_times, galacticus_nodes

subroutine: node_component_merging_statistics_standard_node_merger
Description: Record any major merger of node.
Code lines: 22
Contained by: module node_component_merging_statistics_standard
Modules used: galacticus_nodes

subroutine: `node_component_merging_statistics_standard_node_promotion`
Description: Ensure that `node` is ready for promotion to its parent. In this case, we simply update the node merger time.
Code lines: 20
Contained by: module `node_component_merging_statistics_standard`
Modules used: `galacticus_nodes`

subroutine: `node_component_merging_statistics_standard_reset_hierarchy`
Description: Reset the maximum node hierarchy level if the node has grown sufficiently in mass.
Code lines: 15
Contained by: module `node_component_merging_statistics_standard`
Modules used: `galacticus_nodes`

subroutine: `node_component_merging_statistics_standard_satellite_merging`
Description: Record properties of a merging event for `node`.
Code lines: 23
Contained by: module `node_component_merging_statistics_standard`
Modules used: `galacticus_nodes` `satellite_merging_remnant_properties`

subroutine: `node_component_merging_statistics_standard_thread_initialize`
Description: Initializes the tree node standard merging statistics module.
Code lines: 11
Contained by: module `node_component_merging_statistics_standard`
Modules used: `galacticus_nodes` `input_parameters`

subroutine: `node_component_merging_statistics_standard_thread_uninitialize`
Description: Uninitializes the tree node standard merging statistics module.
Code lines: 9
Contained by: module `node_component_merging_statistics_standard`
Modules used: `galacticus_nodes`

file: `objects.nodes.components.nBody.generic.F90`
Description: Contains a module with the standard implementation of N-body component method.
Code lines: 366

module: `node_component_nbody_generic`
Description: An implementation of the N-body component which supports generic properties.
Code lines: 344
Contained by: file `objects.nodes.components.nBody.generic.F90`
Modules used: `iso_varying_string`
Used by: subroutine `standardpromote`; subroutine `standardoutput`; subroutine `standardpropertiescount`; subroutine `standardpropertynamesestablish`; subroutine `node_components_initialize`

function: `node_component_nbody_generic_add_integer_property`
Description: Add a new real-valued property to this component, and return the index of the property.
Code lines: 39

Contained by: module node_component_nbody_generic
Modules used: galacticus_nodes

function: node_component_nbody_generic_add_real_property
Description: Add a new real-valued property to this component, and return the index of the property.
Code lines: 38
Contained by: module node_component_nbody_generic
Modules used: galacticus_nodes

subroutine: node_component_nbody_generic_initialize
Description: Initializes the generic N-body component module.
Code lines: 17
Contained by: module node_component_nbody_generic
Modules used: galacticus_nodes input_parameters

subroutine: node_component_nbody_generic_output
Description: Store black hole properties in the Galacticus output file buffers.
Code lines: 44
Contained by: module node_component_nbody_generic
Modules used: galacticus_nodes kind_numbers multi_counters

subroutine: node_component_nbody_generic_output_count
Description: Account for the number of black hole properties to be written to the the Galacticus output file.
Code lines: 14
Contained by: module node_component_nbody_generic
Modules used: galacticus_nodes

subroutine: node_component_nbody_generic_output_names
Description: Set names of black hole properties to be written to the Galacticus output file.
Code lines: 33
Contained by: module node_component_nbody_generic
Modules used: galacticus_nodes string_handling

subroutine: node_component_nbody_generic_promote
Description: Ensure that node is ready for promotion to its parent. In this case, we simply update the properties of node to be those of its parent.
Code lines: 22
Contained by: module node_component_nbody_generic
Modules used: galacticus_nodes

subroutine: node_component_nbody_generic_set_integer_property
Description: Set the value of the indexed real property.
Code lines: 18
Contained by: module node_component_nbody_generic
Modules used: galacticus_error galacticus_nodes kind_numbers

subroutine: `node_component_nbody_generic_set_real_property`
Description: Set the value of the indexed real property.
Code lines: 17
Contained by: module `node_component_nbody_generic`
Modules used: `galacticus_error` `galacticus_nodes`

file: `objects.nodes.components.position.preset.F90`
Description: Contains a module which implements a preset position component.
Code lines: 185

module: `node_component_position_preset`
Description: Implements a preset position component.
Code lines: 163
Contained by: file `objects.nodes.components.position.preset.F90`
Used by: subroutine `standardmerge` subroutine `standardpromote`
subroutine `node_components_initialize` subroutine `satellite_move_to_new_host`

subroutine: `node_component_position_preset_initialize`
Code lines: 19
Contained by: module `node_component_position_preset`
Modules used: `galacticus_nodes` `input_parameters`

subroutine: `node_component_position_preset_inter_tree_insert`
Description: A satellite node is being moved between trees, and being added as a new satellite. Its (future-)histories will have been assigned to the `replaceNode` so must be transferred.
Code lines: 29
Contained by: module `node_component_position_preset`
Modules used: `galacticus_nodes` `histories`

subroutine: `node_component_position_preset_move`
Description: Optionally move a satellite to coincide with the postion of its host.
Code lines: 18
Contained by: module `node_component_position_preset`
Modules used: `galacticus_nodes`

subroutine: `node_component_position_preset_node_promotion`
Description: Ensure that `thisNode` is ready for promotion to its parent. In this case, update the position of `thisNode` to that of the parent.
Code lines: 30
Contained by: module `node_component_position_preset`
Modules used: `galacticus_nodes`

file: `objects.nodes.components.position.preset.orphans.F90`
Description: Contains a module which implements a preset position component.
Code lines: 175

module: `node_component_position_preset_orphans`
Description: Implements a preset position component with placement of orphan galaxies.

Code lines: 153
Contained by: file objects.nodes.components.position.preset.orphans.F90
Modules used: satellite_oprhan_distributions
Used by: subroutine node_components_initialize; subroutine node_components_thread_initialize; subroutine node_components_thread_uninitialize

subroutine: node_component_position_preset_orphans_initialize
Code lines: 15
Contained by: module node_component_position_preset_orphans
Modules used: galacticus_nodes; input_parameters

function: node_component_position_preset_orphans_position_orphan
Description: Return the position of the orphan node.
Code lines: 18
Contained by: module node_component_position_preset_orphans
Modules used: galacticus_nodes

subroutine: node_component_position_preset_orphans_thread_initialize
Description: Initializes the tree node preset orphans position module.
Code lines: 11
Contained by: module node_component_position_preset_orphans
Modules used: galacticus_nodes; input_parameters

subroutine: node_component_position_preset_orphans_thread_uninitialize
Description: Uninitializes the tree node preset orphans position module.
Code lines: 9
Contained by: module node_component_position_preset_orphans
Modules used: galacticus_nodes

function: node_component_position_preset_orphans_velocity_orphan
Description: Return the velocity of the orphan node.
Code lines: 18
Contained by: module node_component_position_preset_orphans
Modules used: galacticus_nodes

file: objects.nodes.components.position.trace_dark_matter.F90
Description: Contains a module which implements a preset position component.
Code lines: 146

module: node_component_position_trace_dark_matter
Description: Implements a preset position component.
Code lines: 124
Contained by: file objects.nodes.components.position.trace_dark_matter.F90
Modules used: dark_matter_halo_scales
Used by: subroutine merger_tree_initialize; subroutine standardmerge; subroutine node_components_thread_initialize; subroutine node_components_thread_uninitialize

subroutine satellite_move_to_new_host

subroutine: node_component_position_trace_dark_matter_assign
Description: Assign a position to a satellite.
Code lines: 16
Contained by: module node_component_position_trace_dark_matter
Modules used: galacticus_nodes satellite_oprhan_distributions

subroutine: node_component_position_trace_dark_matter_initialize
Description: Initialize the position of node.
Code lines: 17
Contained by: module node_component_position_trace_dark_matter
Modules used: galacticus_error galacticus_nodes

subroutine: node_component_position_trace_dark_matter_thread_initialize
Description: Initializes the tree node scale dark matter profile module.
Code lines: 11
Contained by: module node_component_position_trace_dark_matter
Modules used: galacticus_nodes input_parameters

subroutine: node_component_position_trace_dark_matter_thread_uninitialize
Description: Uninitializes the tree node scale dark matter profile module.
Code lines: 9
Contained by: module node_component_position_trace_dark_matter
Modules used: galacticus_nodes

subroutine: node_component_position_trace_dark_matter_update
Description: Initialize the position of node.
Code lines: 17
Contained by: module node_component_position_trace_dark_matter
Modules used: galacticus_error galacticus_nodes

file: objects.nodes.components.satellite.orbiting.F90
Description: Contains a module of satellite orbit tree node methods.
Code lines: 529

module: node_component_satellite_orbiting
Description: Implements the orbiting satellite component.
Code lines: 506
Contained by: file objects.nodes.components.satellite.orbiting.F90
Modules used: dark_matter_halo_scales kepler_orbits satellite_dynamical_friction satellite_tidal_heating satellite_tidal_stripping tensors virial_orbits
Used by: subroutine merger_tree_initialize subroutine standardderivativescompute subroutine standardevolve subroutine standardmerge subroutine node_components_thread_initialize subroutine node_components_thread_uninitialize

subroutine evolveforestsperform

subroutine: node_component_satellite_orbiting_bound_mass_initialize
Description: Set the initial bound mass of the satellite.
Code lines: 31
Contained by: module node_component_satellite_orbiting
Modules used: dark_matter_profile_mass_definitions, galactic_structure_enclosed_masses, galacticus_error, galacticus_nodes

subroutine: node_component_satellite_orbiting_create
Description: Create a satellite orbit component and assign initial position, velocity, orbit, and tidal heating quantities. (The initial bound mass is automatically set to the original halo mass by virtue of that being the class default).
Code lines: 41
Contained by: module node_component_satellite_orbiting
Modules used: galacticus_nodes, satellite_merging_timescales

subroutine: node_component_satellite_orbiting_initialize
Description: Initializes the orbiting satellite methods module.
Code lines: 71
Contained by: module node_component_satellite_orbiting
Modules used: galacticus_error, galacticus_nodes, input_parameters, iso_varying_string

subroutine: node_component_satellite_orbiting_rate_compute
Description: Compute rate of change for satellite properties.
Code lines: 109
Contained by: module node_component_satellite_orbiting
Modules used: galactic_structure_densities, galactic_structure_enclosed_masses, galactic_structure_options, galacticus_nodes, numerical_constants_astronomical, numerical_constants_math, numerical_constants_physical, numerical_constants_prefixes, vectors

subroutine: node_component_satellite_orbiting_scale_set
Description: Set scales for properties of thisNode.
Code lines: 30
Contained by: module node_component_satellite_orbiting
Modules used: galactic_structure_enclosed_masses, galacticus_nodes, numerical_constants_astronomical, numerical_constants_prefixes

subroutine: node_component_satellite_orbiting_thread_initialize
Description: Initializes the tree node orbiting satellite module.
Code lines: 15
Contained by: module node_component_satellite_orbiting
Modules used: galacticus_nodes, input_parameters

subroutine: node_component_satellite_orbiting_thread_uninitialize
Description: Uninitializes the tree node orbiting satellite module.

Code lines: 13
Contained by: module node_component_satellite_orbiting
Modules used: galacticus_nodes

subroutine: node_component_satellite_orbiting_tree_initialize
Description: Initialize the orbiting satellite component.
Code lines: 8
Contained by: module node_component_satellite_orbiting
Modules used: galacticus_nodes

subroutine: node_component_satellite_orbiting_trigger_merger
Description: Trigger a merger of the satellite by setting the time until merging to zero.
Code lines: 10
Contained by: module node_component_satellite_orbiting
Modules used: galacticus_nodes

subroutine: node_component_satellite_orbiting_virial_orbit_set
Description: Set the orbit of the satellite at the virial radius.
Code lines: 28
Contained by: module node_component_satellite_orbiting
Modules used: coordinates galacticus_nodes

file: objects.nodes.components.satellite.preset.F90
Description: Contains a module which implements a preset satellite orbit component.
Code lines: 435

module: node_component_satellite_preset
Description: Implements a preset satellite orbit component.
Code lines: 411
Contained by: file objects.nodes.components.satellite.preset.F90
Used by: function node_branch_jump; function node_pull_from_tree; function node_subhalo_promotion; subroutine standardderivativescompute; subroutine standardpromote; subroutine satellite_move_to_new_host

subroutine: node_component_satellite_preset_inter_tree_attach
Description: A satellite node is being moved between trees and attached as the primary progenitor of an existing satellite node. Ensure that preset satellite properties are correctly handled.
Code lines: 54
Contained by: module node_component_satellite_preset
Modules used: galacticus_nodes histories

subroutine: node_component_satellite_preset_inter_tree_insert
Description: A satellite node is being moved between trees, and being added as a new satellite. Its (future-)histories will have been assigned to the replaceNode so must be transferred.
Code lines: 40
Contained by: module node_component_satellite_preset
Modules used: galacticus_nodes histories

subroutine: node_component_satellite_preset_inter_tree_postprocess

Description: For inter-tree node transfers, ensure that any orphaned mergees of the transferred node are transferred over to the new branch.
Code lines: 54
Contained by: module node_component_satellite_preset
Modules used: galacticus_display galacticus_nodes iso_varying_string string_handling

subroutine: node_component_satellite_preset_orphanize
Description: Handle orphanization of a preset satellite component. The satellite should be moved to the branch of its target node.
Code lines: 55
Contained by: module node_component_satellite_preset
Modules used: galacticus_display galacticus_nodes iso_varying_string string_handling

subroutine: node_component_satellite_preset_promote
Description: Ensure that thisNode is ready for promotion to its parent. In this case, we simply copy any preset satellite orbit from the parent.
Code lines: 12
Contained by: module node_component_satellite_preset
Modules used: galacticus_nodes

subroutine: node_component_satellite_preset_rate_compute
Description: Interrupt differential evolution when a preset satellite becomes an orphan.
Code lines: 33
Contained by: module node_component_satellite_preset
Modules used: galacticus_nodes histories

subroutine: node_component_satellite_preset_satellite_host_change
Description: For satellite host changes, if the satellite is an orphan with a merge target ensure it remains in the branch of its merge target.
Code lines: 49
Contained by: module node_component_satellite_preset
Modules used: galacticus_display galacticus_nodes iso_varying_string string_handling

file: objects.nodes.components.satellite.standard.F90
Description: Contains a module of satellite orbit tree node methods.
Code lines: 413

module: node_component_satellite_standard
Description: Implements the standard satellite component.
Code lines: 392
Contained by: file objects.nodes.components.satellite.standard.F90
Modules used: dark_matter_halos_mass_loss_rates kepler_orbits satellite_merging_timescales virial_orbits
Used by: subroutine event_halo_formation subroutine merger_tree_initialize subroutine standardderivativescompute subroutine standardevolve subroutine standardmerge subroutine node_components_initialize

subroutine node_components_thread_-initialize
subroutine node_components_thread_-uninitialize
subroutine satellite_move_to_new_host

subroutine: node_component_satellite_standard_create
Description: Create a satellite orbit component and assign a time until merging and a bound mass equal initially to the total halo mass.
Code lines: 51
Contained by: module node_component_satellite_standard
Modules used: galacticus_nodes

subroutine: node_component_satellite_standard_halo_formation_task
Description: Reset the orbits of satellite galaxies on halo formation events.
Code lines: 22
Contained by: module node_component_satellite_standard
Modules used: galacticus_nodes

subroutine: node_component_satellite_standard_inactive
Description: Set Jacobian zero status for properties of node.
Code lines: 16
Contained by: module node_component_satellite_standard
Modules used: galacticus_nodes stellar_luminosities_structure

subroutine: node_component_satellite_standard_initialize
Description: Initializes the standard satellite orbit component module.
Code lines: 42
Contained by: module node_component_satellite_standard
Modules used: galacticus_nodes input_parameters

subroutine: node_component_satellite_standard_rate_compute
Description: Compute the time until satellite merging rate of change.
Code lines: 30
Contained by: module node_component_satellite_standard
Modules used: galacticus_nodes

subroutine: node_component_satellite_standard_scale_set
Description: Set scales for properties of node.
Code lines: 22
Contained by: module node_component_satellite_standard
Modules used: galacticus_nodes

subroutine: node_component_satellite_standard_thread_initialize
Description: Initializes the tree node standard satellite module.
Code lines: 13
Contained by: module node_component_satellite_standard
Modules used: galacticus_nodes input_parameters

subroutine: node_component_satellite_standard_thread_uninitialize
Description: Uninitializes the tree node standard satellite module.

Code lines: 11
Contained by: module node_component_satellite_standard
Modules used: galacticus_nodes

subroutine: node_component_satellite_standard_tree_initialize
Description: Initialize the standard satellite component.
Code lines: 8
Contained by: module node_component_satellite_standard
Modules used: galacticus_nodes

function: node_component_satellite_standard_virial_orbit
Description: Return the orbit of the satellite at the virial radius.
Code lines: 29
Contained by: module node_component_satellite_standard
Modules used: galacticus_nodes

subroutine: node_component_satellite_standard_virial_orbit_set
Description: Set the orbit of the satellite at the virial radius.
Code lines: 24
Contained by: module node_component_satellite_standard
Modules used: galacticus_nodes

file: objects.nodes.components.satellite.very_simple.F90
Description: Contains a module which implements a very simple satellite orbit component.
Code lines: 239

module: node_component_satellite_very_simple
Description: Implements a very simple satellite orbit component.
Code lines: 217
Contained by: file objects.nodes.components.satellite.very_simple.F90
Modules used: satellite_merging_timescales, virial_orbits
Used by: subroutine event_halo_formation, subroutine merger_tree_initialize, subroutine standardderivativescompute, subroutine standardmerge, subroutine node_components_initialize, subroutine node_components_thread_initialize, subroutine node_components_thread_uninitialize, subroutine satellite_move_to_new_host

subroutine: node_component_satellite_very_simple_create
Description: Create a satellite orbit component and assign a time until merging and a bound mass equal initially to the total halo mass.
Code lines: 41
Contained by: module node_component_satellite_very_simple
Modules used: galacticus_nodes, kepler_orbits

subroutine: node_component_satellite_very_simple_halo_formation_task
Description: Reset the orbits of satellite galaxies on halo formation events.
Code lines: 21
Contained by: module node_component_satellite_very_simple

Modules used: galacticus_nodes

subroutine: node_component_satellite_very_simple_initialize
Description: Initializes the tree node satellite orbit methods module.
Code lines: 16
Contained by: module node_component_satellite_very_simple
Modules used: input_parameters

subroutine: node_component_satellite_very_simple_rate_compute
Description: Compute the time until satellite merging rate of change.
Code lines: 24
Contained by: module node_component_satellite_very_simple
Modules used: galacticus_nodes

subroutine: node_component_satellite_very_simple_thread_initialize
Description: Initializes the tree node very simple satellite module.
Code lines: 12
Contained by: module node_component_satellite_very_simple
Modules used: galacticus_nodes input_parameters

subroutine: node_component_satellite_very_simple_thread_uninitialize
Description: Uninitializes the tree node very simple satellite module.
Code lines: 10
Contained by: module node_component_satellite_very_simple
Modules used: galacticus_nodes

subroutine: node_component_satellite_very_simple_tree_initialize
Description: Initialize the very simple satellite component.
Code lines: 8
Contained by: module node_component_satellite_very_simple
Modules used: galacticus_nodes

file: objects.nodes.components.spheroid.standard.F90
Description: Contains a module which implements the standard spheroid component.
Code lines: 1467

module: node_component_spheroid_standard
Description: Implements the standard spheroid component.
Code lines: 1445
Contained by: file objects.nodes.components.spheroid.standard.F90
Modules used: dark_matter_halo_scales, histories, iso_varying_string, node_component_spheroid_standard_data, ram_pressure_stripping_mass_loss_-rate_spheroids, satellites_tidal_fields, star_formation_feedback_expulsion_-spheroids, star_formation_feedback_spheroids, star_formation_histories, star_formation_timescales_spheroids, stellar_population_properties, tidal_stripping_mass_loss_rate_-spheroids

Used by: subroutine equilibriumsolve; subroutine fixedsolve; subroutine linearsolve; subroutine simplesolve; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; subroutine satellitemergerprocess; subroutine standardderivativescompute; subroutine standardevolve; subroutine standardpoststepprocessing; subroutine node_components_initialize; subroutine node_components_thread_initialize; subroutine node_components_thread_uninitialize; subroutine evolveforestsperform

subroutine: node_component_spheroid_standard_energy_gas_input_rate

Description: Handles input of energy into the spheroid gas from other components (e.g. black holes). The energy input rate should be in units of $M_\odot$ km^2 s^{-2} Gyr^{-1}.
Code lines: 46
Contained by: module node_component_spheroid_standard
Modules used: abundances_structure; galacticus_error; galacticus_nodes

subroutine: node_component_spheroid_standard_inactive

Description: Set Jacobian zero status for properties of node.
Code lines: 16
Contained by: module node_component_spheroid_standard
Modules used: galacticus_nodes; stellar_luminosities_structure

subroutine: node_component_spheroid_standard_initialize

Description: Initializes the tree node standard spheroid methods module.
Code lines: 67
Contained by: module node_component_spheroid_standard
Modules used: abundances_structure; galacticus_error; galacticus_nodes; input_parameters

subroutine: node_component_spheroid_standard_initializor

Description: Initializes a standard spheroid component.
Code lines: 45
Contained by: module node_component_spheroid_standard
Modules used: galacticus_nodes; histories

subroutine: node_component_spheroid_standard_mass_gas_sink_rate

Description: Account for a sink of gaseous material in the standard spheroid.
Code lines: 28
Contained by: module node_component_spheroid_standard
Modules used: abundances_structure; galacticus_error; galacticus_nodes

subroutine: node_component_spheroid_standard_post_evolve

Description: Trim histories attached to the spheroid.
Code lines: 23
Contained by: module node_component_spheroid_standard

Modules used: galacticus_nodes histories stellar_luminosities_structure

subroutine: node_component_spheroid_standard_post_step
Description: Trim histories attached to the spheroid.
Code lines: 113
Contained by: module node_component_spheroid_standard
Modules used: abundances_structure fgsl galacticus_display galacticus_error galacticus_nodes iso_varying_string stellar_luminosities_structure string_handling

subroutine: node_component_spheroid_standard_pre_evolve
Description: Ensure the spheroid has been initialized.
Code lines: 16
Contained by: module node_component_spheroid_standard
Modules used: galacticus_nodes

function: node_component_spheroid_standard_radius_solve
Description: Return the circular radius of the standard spheroid.
Code lines: 10
Contained by: module node_component_spheroid_standard
Modules used: galacticus_nodes

subroutine: node_component_spheroid_standard_radius_solve_set
Description: Set the scale radius of the standard spheroid.
Code lines: 11
Contained by: module node_component_spheroid_standard
Modules used: galacticus_nodes

subroutine: node_component_spheroid_standard_radius_solver
Description: Interface for the size solver algorithm.
Code lines: 44
Contained by: module node_component_spheroid_standard
Modules used: galacticus_nodes

subroutine: node_component_spheroid_standard_radius_solver_plausibility
Description: Determines whether the spheroid is physically plausible for radius solving tasks. Require that it have non-zero mass and angular momentum.
Code lines: 37
Contained by: module node_component_spheroid_standard
Modules used: dark_matter_halo_scales galacticus_nodes

subroutine: node_component_spheroid_standard_rate_compute
Description: Compute the standard spheroid node mass rate of change.
Code lines: 170
Contained by: module node_component_spheroid_standard
Modules used: abundances_structure galactic_structure_options galacticus_error galacticus_nodes

histories numerical_constants_astronomical
stellar_luminosities_structure

subroutine: node_component_spheroid_standard_satellite_merging
Description: Transfer any standard spheroid associated with node to its host halo.
Code lines: 209
Contained by: module node_component_spheroid_standard
Modules used: abundances_structure galacticus_error
galacticus_nodes satellite_merging_mass_movements
satellite_merging_remnant_properties satellite_merging_remnant_sizes
stellar_luminosities_structure

subroutine: node_component_spheroid_standard_scale_set
Description: Set scales for properties of node. Note that gas masses get an additional scaling down since they can approach zero and we'd like to prevent them from becoming negative.
Code lines: 62
Contained by: module node_component_spheroid_standard
Modules used: abundances_structure galacticus_nodes
histories stellar_luminosities_structure

subroutine: node_component_spheroid_standard_star_formation_history_extend
Description: Extend the range of a star formation history in a standard spheroid component for node.
Code lines: 15
Contained by: module node_component_spheroid_standard
Modules used: galacticus_nodes

subroutine: node_component_spheroid_standard_star_formation_history_output
Description: Store the star formation history in the output file.
Code lines: 23
Contained by: module node_component_spheroid_standard
Modules used: galacticus_nodes histories
iso_c_binding kind_numbers

subroutine: node_component_spheroid_standard_star_formation_history_rate
Description: Adjust the rates for the star formation history.
Code lines: 32
Contained by: module node_component_spheroid_standard
Modules used: galacticus_error galacticus_nodes
memory_management

function: node_component_spheroid_standard_star_formation_rate
Description: Return the star formation rate of the standard spheroid.
Code lines: 26
Contained by: module node_component_spheroid_standard
Modules used: galacticus_nodes

subroutine: node_component_spheroid_standard_state_retrieve
Description: Retrieve the tabulation state from the file.
Code lines: 20

Contained by: module node_component_spheroid_standard
Modules used: fgsl galacticus_display galacticus_error iso_c_binding

subroutine: node_component_spheroid_standard_state_store
Description: Write the tablulation state to file.
Code lines: 15
Contained by: module node_component_spheroid_standard
Modules used: fgsl galacticus_display iso_c_binding

subroutine: node_component_spheroid_standard_stellar_prprts_history_extend
Description: Extend the range of a stellar properties history in a standard spheroid component for node.
Code lines: 15
Contained by: module node_component_spheroid_standard
Modules used: galacticus_nodes

subroutine: node_component_spheroid_standard_stellar_prprts_history_rate
Description: Adjust the rates for the stellar properties history.
Code lines: 32
Contained by: module node_component_spheroid_standard
Modules used: galacticus_error galacticus_nodes memory_management

subroutine: node_component_spheroid_standard_thread_initialize
Description: Initializes the standard spheroid module for each thread.
Code lines: 56
Contained by: module node_component_spheroid_standard
Modules used: galacticus_error galacticus_nodes input_parameters

subroutine: node_component_spheroid_standard_thread_uninitialize
Description: Uninitializes the standard spheroid module for each thread.
Code lines: 18
Contained by: module node_component_spheroid_standard
Modules used: galacticus_nodes

function: node_component_spheroid_standard_velocity_solve
Description: Return the circular velocity of the standard spheroid.
Code lines: 10
Contained by: module node_component_spheroid_standard
Modules used: galacticus_nodes

subroutine: node_component_spheroid_standard_velocity_solve_set
Description: Set the scale velocity of the standard spheroid.
Code lines: 11
Contained by: module node_component_spheroid_standard
Modules used: galacticus_nodes

file: objects.nodes.components.spheroid.standard.data.F90
Description: Contains a module of data for standard spheroid components.
Code lines: 32

module: node_component_spheroid_standard_data
Description: Contains data for standard spheroid components.
Code lines: 10
Contained by: file objects.nodes.components.spheroid.standard.data.F90
Modules used: mass_distributions
Used by: function node_component_spheroid_standard_density; function node_component_spheroid_standard_enclosed_mass; function node_component_spheroid_standard_half_mass_radius; function node_component_spheroid_standard_potential; module node_component_spheroid_standard

file: objects.nodes.components.spheroid.very_simple.F90
Description: Contains a module that implements a very simple spheroid component.
Code lines: 821

module: node_component_spheroid_very_simple
Description: Implements a very simple spheroid component.
Code lines: 799
Contained by: file objects.nodes.components.spheroid.very_simple.F90
Modules used: dark_matter_halo_scales; dark_matter_profiles_dmo; iso_varying_string; star_formation_feedback_spheroids; star_formation_timescales_spheroids; stellar_population_properties
Used by: subroutine equilibriumsolve; subroutine fixedsolve; subroutine linearsolve; subroutine simplesolve; subroutine satellitemergerprocess; subroutine standardderivativescompute; subroutine standardevolve; subroutine standardpoststepprocessing; subroutine node_components_initialize; subroutine node_components_thread_initialize; subroutine node_components_thread_uninitialize

subroutine: node_component_spheroid_very_simple_create
Description: Create properties in a very simple spheroid component.
Code lines: 27
Contained by: module node_component_spheroid_very_simple
Modules used: galacticus_nodes; histories

subroutine: node_component_spheroid_very_simple_initialize
Description: Initializes the tree node very simple spheroid component module.
Code lines: 63
Contained by: module node_component_spheroid_very_simple
Modules used: galacticus_nodes; input_parameters

subroutine: `node_component_spheroid_very_simple_post_evolve`
Description: Catch rounding errors in the very simple spheroid gas evolution.
Code lines: 23
Contained by: module `node_component_spheroid_very_simple`
Modules used: `galacticus_nodes` `histories` `stellar_luminosities_structure`

subroutine: `node_component_spheroid_very_simple_post_step`
Description: Catch rounding errors in the very simple spheroid gas evolution.
Code lines: 63
Contained by: module `node_component_spheroid_very_simple`
Modules used: `abundances_structure` `fgsl` `galacticus_display` `galacticus_nodes` `stellar_luminosities_structure` `string_handling`

subroutine: `node_component_spheroid_very_simple_pre_evolve`
Description: Ensure the spheroid has been initialized.
Code lines: 16
Contained by: module `node_component_spheroid_very_simple`
Modules used: `galacticus_nodes`

function: `node_component_spheroid_very_simple_radius`
Description: Return the radius of the spheroid used in structure solvers.
Code lines: 10
Contained by: module `node_component_spheroid_very_simple`
Modules used: `galacticus_nodes`

subroutine: `node_component_spheroid_very_simple_radius_set`
Description: Set the radius of the spheroid used in structure solvers.
Code lines: 11
Contained by: module `node_component_spheroid_very_simple`
Modules used: `galacticus_nodes`

subroutine: `node_component_spheroid_very_simple_radius_solver`
Description: Interface for the size solver algorithm.
Code lines: 31
Contained by: module `node_component_spheroid_very_simple`
Modules used: `dark_matter_halo_spins` `galacticus_nodes`

subroutine: `node_component_spheroid_very_simple_radius_solver_plausibility`
Description: Determines whether the spheroid is physically plausible for radius solving tasks. Require that it have non-zero mass.
Code lines: 16
Contained by: module `node_component_spheroid_very_simple`
Modules used: `galacticus_nodes`

subroutine: `node_component_spheroid_very_simple_rate_compute`
Description: Compute the very simple spheroid node mass rate of change.

Code lines: 70
Contained by: module node_component_spheroid_very_simple
Modules used: abundances_structure, galactic_structure_options, galacticus_nodes, histories, stellar_feedback, stellar_luminosities_structure

subroutine: node_component_spheroid_very_simple_rates
Description: Compute rates.
Code lines: 48
Contained by: module node_component_spheroid_very_simple
Modules used: abundances_structure, galactic_structure_options, galacticus_nodes, histories, stellar_feedback, stellar_luminosities_structure

subroutine: node_component_spheroid_very_simple_satellite_merging
Description: Transfer any very simple spheroid associated with node to its host halo.
Code lines: 129
Contained by: module node_component_spheroid_very_simple
Modules used: abundances_structure, galacticus_error, galacticus_nodes, histories, satellite_merging_mass_movements, satellite_merging_remnant_properties, stellar_luminosities_structure

subroutine: node_component_spheroid_very_simple_scale_set
Description: Set scales for properties of node.
Code lines: 39
Contained by: module node_component_spheroid_very_simple
Modules used: abundances_structure, galacticus_nodes, histories, stellar_luminosities_structure

function: node_component_spheroid_very_simple_sfr
Description: Return the star formation rate of the very simple spheroid.
Code lines: 21
Contained by: module node_component_spheroid_very_simple
Modules used: galacticus_nodes

subroutine: node_component_spheroid_very_simple_thread_initialize
Description: Initializes the tree node very simple satellite module.
Code lines: 15
Contained by: module node_component_spheroid_very_simple
Modules used: galacticus_nodes, input_parameters

subroutine: node_component_spheroid_very_simple_thread_uninitialize
Description: Uninitializes the tree node very simple satellite module.
Code lines: 13
Contained by: module node_component_spheroid_very_simple
Modules used: galacticus_nodes

function: `node_component_spheroid_very_simple_velocity`
Description: Return the circular velocity of the spheroid.
Code lines: 10
Contained by: module `node_component_spheroid_very_simple`
Modules used: `galacticus_nodes`

subroutine: `node_component_spheroid_very_simple_velocity_set`
Description: Set the circular velocity of the spheroid.
Code lines: 11
Contained by: module `node_component_spheroid_very_simple`
Modules used: `galacticus_nodes`

file: `objects.nodes.components.spin.Vitvitska.F90`
Description: Contains a module implementing a node spin component using the approach of Vitvitska et al. [2002].
Code lines: 402

module: `node_component_spin_vitvitska`
Description: Implements a node spin component using the approach of Vitvitska et al. [2002].
Code lines: 377
Contained by: file `objects.nodes.components.spin.Vitvitska.F90`
Modules used: `dark_matter_halo_scales`, `dark_matter_halo_spins`, `dark_matter_profile_scales`, `dark_matter_profiles_dmo`, `halo_spin_distributions`, `iso_varying_string`, `kepler_orbits`, `numerical_constants_physical`, `vectors`, `virial_orbits`
Used by: subroutine `merger_tree_initialize`; subroutine `standardderivativescompute`; subroutine `standardevolve`; subroutine `standardpromote`; subroutine `node_components_initialize`; subroutine `node_components_thread_initialize`; subroutine `node_components_thread_uninitialize`

subroutine: `node_component_spin_vitvitska_bindings`
Description: Initializes the "Vitvitskae" implementation of the spin component.
Code lines: 21
Contained by: module `node_component_spin_vitvitska`
Modules used: `galacticus_nodes`, `input_parameters`

subroutine: `node_component_spin_vitvitska_branch_initialize`
Description: Ensure a branch of a merger tree is initialized with spins in the Vitvitska model.
Code lines: 14
Contained by: module `node_component_spin_vitvitska`
Modules used: `galacticus_nodes`, `merger_tree_walkers`

subroutine: `node_component_spin_vitvitska_initialize_spins`
Description: Initialize the spin of `node`.
Code lines: 76

Contained by: module node_component_spin_vitvitska
Modules used: galacticus_nodes iso_varying_string

subroutine: node_component_spin_vitvitska_promote
Description: Ensure that node is ready for promotion to its parent. In this case, we simply update the spin of node to be consistent with the merging event.
Code lines: 26
Contained by: module node_component_spin_vitvitska
Modules used: galacticus_error galacticus_nodes

subroutine: node_component_spin_vitvitska_rate_compute
Description: Compute rates of change of properties in the Vitvitska implementation of the spin component.
Code lines: 33
Contained by: module node_component_spin_vitvitska
Modules used: galacticus_nodes

subroutine: node_component_spin_vitvitska_scale_set
Description: Set scales for properties in the Vitvitska implementation of the spin component.
Code lines: 17
Contained by: module node_component_spin_vitvitska
Modules used: galacticus_nodes

function: node_component_spin_vitvitska_spin
Description: Return the spin parameter.
Code lines: 9
Contained by: module node_component_spin_vitvitska
Modules used: galacticus_nodes

function: node_component_spin_vitvitska_spin_growth_rate
Description: Return the growth rate of the spin parameter.
Code lines: 12
Contained by: module node_component_spin_vitvitska
Modules used: galacticus_nodes

function: node_component_spin_vitvitska_spin_vector
Description: Return the spin parameter vector.
Code lines: 10
Contained by: module node_component_spin_vitvitska
Modules used: galacticus_nodes

subroutine: node_component_spin_vitvitska_thread_initialize
Description: Initializes the tree node Vitvitsake spin module.
Code lines: 16
Contained by: module node_component_spin_vitvitska
Modules used: dark_matter_profile_scales galacticus_nodes input_parameters

subroutine: node_component_spin_vitvitska_thread_uninitialize

Description: Uninitializes the tree node Vitvitsake spin module.
Code lines: 13
Contained by: module node_component_spin_vitvitska
Modules used: galacticus_nodes

function: orbital_angular_momentum
Description: Returns the orbital angular momentum vector associated with a satellite by drawing a random position towards the host at virial radius distance and a random velocity vector consistent with the orbital parameters of the satellite.
Code lines: 26
Contained by: module node_component_spin_vitvitska
Modules used: coordinates galacticus_nodes

file: objects.nodes.components.spin.preset.F90
Description: Contains a module which implements the preset spin component.
Code lines: 169

module: node_component_spin_preset
Description: Implements the preset spin component.
Code lines: 147
Contained by: file objects.nodes.components.spin.preset.F90
Used by: subroutine merger_tree_initialize subroutine standardderivativescompute subroutine standardevolve subroutine standardpromote

subroutine: node_component_spin_preset_initialize
Description: Initialize the spin of node.
Code lines: 31
Contained by: module node_component_spin_preset
Modules used: galacticus_nodes

subroutine: node_component_spin_preset_promote
Description: Ensure that node is ready for promotion to its parent. In this case, we simply update the spin of node to be that of its parent.
Code lines: 25
Contained by: module node_component_spin_preset
Modules used: galacticus_error galacticus_nodes

subroutine: node_component_spin_preset_rate_compute
Description: Compute rates of change of properties in the preset implementation of the spin component.
Code lines: 25
Contained by: module node_component_spin_preset
Modules used: galacticus_nodes

subroutine: node_component_spin_preset_scale_set
Description: Set scales for properties in the preset implementation of the spin component.
Code lines: 16
Contained by: module node_component_spin_preset
Modules used: galacticus_nodes

file: `objects.nodes.components.spin.preset3D.F90`
Description: Contains a module which implements the preset 3-D spin component.
Code lines: 174

module: `node_component_spin_preset3d`
Description: Implements the preset spin component.
Code lines: 152
Contained by: file `objects.nodes.components.spin.preset3D.F90`
Used by: subroutine `merger_tree_initialize`; subroutine `standardderivativescompute`; subroutine `standardevolve`; subroutine `standardpromote`

subroutine: `node_component_spin_preset3d_initialize`
Description: Initialize the spin of `node`.
Code lines: 31
Contained by: module `node_component_spin_preset3d`
Modules used: `galacticus_nodes`

subroutine: `node_component_spin_preset3d_promote`
Description: Ensure that `node` is ready for promotion to its parent. In this case, we simply update the spin of `node` to be that of its parent.
Code lines: 25
Contained by: module `node_component_spin_preset3d`
Modules used: `galacticus_error`; `galacticus_nodes`

subroutine: `node_component_spin_preset3d_rate_compute`
Description: Compute rates of change of properties in the preset implementation of the spin component.
Code lines: 25
Contained by: module `node_component_spin_preset3d`
Modules used: `galacticus_nodes`

subroutine: `node_component_spin_preset3d_scale_set`
Description: Set scales for properties in the preset implementation of the spin component.
Code lines: 16
Contained by: module `node_component_spin_preset3d`
Modules used: `galacticus_nodes`

file: `objects.nodes.components.spin.random.F90`
Description: Contains a module of spin tree node methods.
Code lines: 189

module: `node_component_spin_random`
Description: Implement random spin tree node method.
Code lines: 167
Contained by: file `objects.nodes.components.spin.random.F90`
Modules used: `halo_spin_distributions`
Used by: subroutine `merger_tree_initialize`; subroutine `standardpromote`; subroutine `node_components_initialize`; subroutine `node_components_thread_initialize`

subroutine node_components_thread_-
uninitialize

subroutine: node_component_spin_random_initialize
Description: Initializes the random spin component module.
Code lines: 15
Contained by: module node_component_spin_random
Modules used: input_parameters

subroutine: node_component_spin_random_initialize_spins
Description: Initialize the spin of node.
Code lines: 43
Contained by: module node_component_spin_random
Modules used: galacticus_nodes

subroutine: node_component_spin_random_promote
Description: Ensure that node is ready for promotion to its parent. In this case, we simply update the spin of node to be that of its parent.
Code lines: 24
Contained by: module node_component_spin_random
Modules used: galacticus_error galacticus_nodes

subroutine: node_component_spin_random_thread_initialize
Description: Initializes the tree node random spin module.
Code lines: 11
Contained by: module node_component_spin_random
Modules used: galacticus_nodes input_parameters

subroutine: node_component_spin_random_thread_uninitialize
Description: Uninitializes the tree node random spin module.
Code lines: 9
Contained by: module node_component_spin_random
Modules used: galacticus_nodes

file: objects.stellar_luminosities.F90
Description: Contains a module which defines the stellar luminosities object.
Code lines: 1671

module: stellar_luminosities_structure
Description: Defines the stellar luminosities object.
Code lines: 1647
Contained by: file objects.stellar_luminosities.F90
Modules used: iso_varying_string stellar_population_spectra_-
postprocess
Used by: function function
stellarabsolutemagnitudesconstructorinternal stellarabsolutemagnitudesconstructorparameters
function function
stellarabsolutemagnitudespasses stellarapparentmagnitudesconstructorinternal

function stellarapparentmagnitudesconstructorparameters
function stellarapparentmagnitudespasses
subroutine galactic_structure_radii_definition_decode
subroutine galacticus_state_retrieve
subroutine galacticus_state_store

function radiihalflightpropertieselementcount
function radiihalflightpropertiesdescriptions
function radiihalflightpropertiesnames
function radiihalflightpropertiesextract
function radiihalflightpropertiesunitsinsi

function lmnstyemssnlineconstructorinternal
function lmnstyemssnlineextract
function luminositystellarconstructorinternal
function lmnstystllrchrltfll2000constructorinternal
function lmnstystllrchrltfll2000extract
module galacticus_nodes
subroutine node_component_age_statistics_standard_inactive
subroutine node_component_disk_standard_inactive
subroutine node_component_disk_standard_post_evolve
subroutine node_component_disk_standard_post_step
subroutine node_component_disk_standard_rate_compute
subroutine node_component_disk_standard_satellite_merging
subroutine node_component_disk_standard_scale_set
subroutine node_component_disk_very_simple_analytic_solver
subroutine node_component_disk_very_simple_post_evolve
subroutine node_component_disk_very_simple_post_step
subroutine node_component_disk_very_simple_rate_compute
subroutine node_component_disk_very_simple_rates
subroutine node_component_disk_very_simple_scale_set
subroutine node_component_satellite_standard_inactive
subroutine node_component_spheroid_standard_inactive
subroutine node_component_spheroid_standard_post_evolve
subroutine node_component_spheroid_standard_post_step
subroutine node_component_spheroid_standard_rate_compute
subroutine node_component_spheroid_standard_satellite_merging
subroutine node_component_spheroid_standard_scale_set
subroutine node_component_spheroid_very_simple_post_evolve
subroutine node_component_spheroid_very_simple_post_step
subroutine node_component_spheroid_very_simple_rate_compute
subroutine node_component_spheroid_very_simple_rates
subroutine node_component_spheroid_very_simple_satellite_merging
subroutine node_component_spheroid_very_simple_scale_set
file stellar_populations.properties.instantaneous.F90
subroutine noninstantaneousrates

interface: `abs`
Code lines: 2
Contained by: module `stellar_luminosities_structure`

interface: `max`
Code lines: 2
Contained by: module `stellar_luminosities_structure`

interface: `operator(*)`
Code lines: 2
Contained by: module `stellar_luminosities_structure`

subroutine: `sortbyindexpostprocessor`
Description: Given an `array`, sort it in place using the supplied index.
Code lines: 14
Contained by: module `stellar_luminosities_structure`
Modules used: `iso_c_binding`

function: `stellar_luminosities_add`
Description: Add two stellar luminosities objects.
Code lines: 21
Contained by: module `stellar_luminosities_structure`

subroutine: `stellar_luminosities_builder`
Description: Build a `stellarLuminosities` object from the given XML `stellarLuminositiesDefinition`.
Code lines: 20
Contained by: module `stellar_luminosities_structure`
Modules used: `fox_dom` `galacticus_error`

subroutine: `stellar_luminosities_create`
Description: Ensure that the `luminosity` array in a `stellarLuminosities` is allocated.
Code lines: 8
Contained by: module `stellar_luminosities_structure`
Modules used: `memory_management`

subroutine: `stellar_luminosities_deserialize`
Description: Pack stellar luminosities from an array into a `stellarLuminosities` structure.
Code lines: 16
Contained by: module `stellar_luminosities_structure`

subroutine: `stellar_luminosities_destroy`
Description: Destroy an stellarLuminosities object.
Code lines: 7
Contained by: module `stellar_luminosities_structure`

function: `stellar_luminosities_divide`
Description: Divide a stellar luminosities object by a scalar.
Code lines: 11

Contained by: module stellar_luminosities_structure

subroutine: stellar_luminosities_dump
Description: Dump a stellar luminosities object.
Code lines: 24
Contained by: module stellar_luminosities_structure
Modules used: galacticus_display

subroutine: stellar_luminosities_dump_raw
Description: Dump an stellarLuminosities object to binary.
Code lines: 14
Contained by: module stellar_luminosities_structure

subroutine: stellar_luminosities_expand_filter_set
Description: Expand the filter set by removing the filter at index expandFrom by adding expandCount replicas of the filter at that point.
Code lines: 55
Contained by: module stellar_luminosities_structure
Modules used: iso_c_binding memory_management

subroutine: stellar_luminosities_increment
Description: Increment a stellar luminosities object.
Code lines: 17
Contained by: module stellar_luminosities_structure

function: stellar_luminosities_index_from_name
Description: Return the index of and specified entry in the luminosity list given its name.
Code lines: 17
Contained by: module stellar_luminosities_structure
Modules used: galacticus_error

function: stellar_luminosities_index_from_properties
Description: Return the index of and specified entry in the luminosity list given its properties.
Code lines: 46
Contained by: module stellar_luminosities_structure
Modules used: galacticus_error iso_varying_string
numerical_comparison string_handling

subroutine: stellar_luminosities_initialize
Description: Initialize the stellarLuminositiesStructure object module. Determines which stellar luminosities are to be tracked.
Code lines: 207
Contained by: module stellar_luminosities_structure
Modules used: array_utilities cosmology_functions
galacticus_error hii_region_emission_lines
input_parameters instruments_filters
iso_c_binding memory_management
sort

function: `stellar_luminosities_is_output`
Description: Return true or false depending on whether `luminosityIndex` should be output at `time`.
Code lines: 30
Contained by: module `stellar_luminosities_structure`
Modules used: `galacticus_error`

function: `stellar_luminosities_is_zero`
Description: Test whether an stellarLuminosities object is zero.
Code lines: 13
Contained by: module `stellar_luminosities_structure`

function: `stellar_luminosities_luminosity`
Description: Return the requested luminosity from a `stellarLuminosities` object.
Code lines: 21
Contained by: module `stellar_luminosities_structure`
Modules used: `galacticus_error`

function: `stellar_luminosities_multiply`
Description: Multiply a stellar luminosities object by a scalar.
Code lines: 11
Contained by: module `stellar_luminosities_structure`

function: `stellar_luminosities_multiply_switched`
Description: Multiply a stellar luminosities object by a scalar.
Code lines: 11
Contained by: module `stellar_luminosities_structure`

function: `stellar_luminosities_name`
Description: Return a name for the specified entry in the stellar luminosities structure.
Code lines: 16
Contained by: module `stellar_luminosities_structure`
Modules used: `galacticus_error`

function: `stellar_luminosities_non_static_size_of`
Description: Return the size of any non-static components of the object.
Code lines: 13
Contained by: module `stellar_luminosities_structure`
Modules used: `iso_c_binding`

subroutine: `stellar_luminosities_output`
Description: Store a `stellarLuminosities` object in the output buffers.
Code lines: 26
Contained by: module `stellar_luminosities_structure`
Modules used: `kind_numbers` `memory_management` `multi_counters`

subroutine: `stellar_luminosities_output_count`
Description: Increment the output count to account for a `stellarLuminosities` object.

Code lines: 12
Contained by: module stellar_luminosities_structure

function: stellar_luminosities_output_count_get
Description: Compute the number of luminosities to be output at a given time.
Code lines: 13
Contained by: module stellar_luminosities_structure

subroutine: stellar_luminosities_output_names
Description: Assign names to output buffers for a stellarLuminosities object.
Code lines: 26
Contained by: module stellar_luminosities_structure

interface: stellar_luminosities_parameter_map
Code lines: 2
Contained by: module stellar_luminosities_structure

subroutine: stellar_luminosities_parameter_map_double
Description: Map an array of luminosity-related input parameters into a new array accounting for special case processing.
Code lines: 20
Contained by: module stellar_luminosities_structure
Modules used: memory_management

subroutine: stellar_luminosities_post_output
Description: Clean up a stellarLuminosities object after output.
Code lines: 28
Contained by: module stellar_luminosities_structure

function: stellar_luminosities_property_count
Description: Return the number of properties required to track stellar luminosities.
Code lines: 14
Contained by: module stellar_luminosities_structure

subroutine: stellar_luminosities_read_raw
Description: Read an stellarLuminosities object from binary.
Code lines: 19
Contained by: module stellar_luminosities_structure
Modules used: memory_management

subroutine: stellar_luminosities_reset
Description: Reset an stellarLuminosities object.
Code lines: 12
Contained by: module stellar_luminosities_structure

subroutine: stellar_luminosities_sed_top_hat_step
Description: Given a top hat filter central wavelength and filter width, determine the position and width of the next top hat filter in the array.
Code lines: 43
Contained by: module stellar_luminosities_structure

Modules used: stellar_population_spectra

subroutine: `stellar_luminosities_serialize`
Description: Unpack stellar luminosities from a `stellarLuminosities` structure into an array.
Code lines: 15
Contained by: module stellar_luminosities_structure

function: `stellar_luminosities_serialize_count`
Description: Return the number of properties required to track stellar luminosities.
Code lines: 11
Contained by: module stellar_luminosities_structure

subroutine: `stellar_luminosities_set`
Description: Set the luminosity in each band for a single `stellarPopulation_` of given `mass` with the specified `abundancesStellar` and which formed at cosmological `time`.
Code lines: 32
Contained by: module stellar_luminosities_structure
Modules used: abundances_structure, stellar_population_luminosities, stellar_populations

subroutine: `stellar_luminosities_set_to_unity`
Description: Set an stellarLuminosities object to unity.
Code lines: 12
Contained by: module stellar_luminosities_structure

subroutine: `stellar_luminosities_special_cases`
Description: Modify the input list of luminosities for special cases.
Code lines: 232
Contained by: module stellar_luminosities_structure
Modules used: cosmology_functions, hii_region_emission_lines, iso_c_binding, memory_management, output_times, stellar_population_spectra, string_handling

subroutine: `stellar_luminosities_state_restore`
Description: Retrieve the luminosities state from the file.
Code lines: 63
Contained by: module stellar_luminosities_structure
Modules used: fgsl, galacticus_display, instruments_filters, iso_c_binding, iso_varying_string, stellar_population_spectra_postprocess, string_handling

subroutine: `stellar_luminosities_state_store`
Description: Write the luminosities state to file.
Code lines: 31
Contained by: module stellar_luminosities_structure
Modules used: fgsl, galacticus_display

iso_c_binding iso_varying_string
string_handling

function: stellar_luminosities_subtract
Description: Subtract two stellar luminosities objects.
Code lines: 21
Contained by: module stellar_luminosities_structure

subroutine: stellar_luminosities_truncate
Description: Truncate (or pad) the stellar luminosities to match the number in the given templateLuminosities.
Code lines: 32
Contained by: module stellar_luminosities_structure

type: stellarluminosities
Description: The stellar luminosities structure.
Code lines: 217
Contained by: module stellar_luminosities_structure

function: stellarluminositiesabs
Description: Return an element-by-element abs() on a stellar luminosity object.
Code lines: 8
Contained by: module stellar_luminosities_structure

function: stellarluminositiescountmaximum
Code lines: 12
Contained by: module stellar_luminosities_structure

function: stellarluminositiesmax
Description: Return an element-by-element max() on two stellar luminosity objects.
Code lines: 14
Contained by: module stellar_luminosities_structure

file: objects.tables.F90
Description: Contains a module which defines a table class with optimized interpolation operators.
Code lines: 2358

module: tables
Description: Defines a table class with optimized interpolation operators.
Code lines: 2336
Contained by: file objects.tables.F90
Modules used: fgsl table_labels
Used by: file accretion_disks.ADAF.F90 file dark_matter_halos.scales.virial_density_contrast.F90
file dark_matter_halos.spins.distributions.Bett2007.F90 file dark_matter_profiles.structure.concentration.Klypin2015.F90
file dark_matter_profiles_DMO.Burkert.F90 file dark_matter_profiles_DMO.Einasto.F90

file dark_matter_profiles_DMO.NFW.F90

file galacticus.output.analyses.distribution_operator.disk_size_inclination.F90

module galacticus_state

file halo_model.power_spectrum_modifier.triaxiality.F90

file hot_halo.mass_distribution.cored.core_radius.growing.F90

subroutine interface_fsps_ssps_tabulate

module intergalactic_medium_state

file mass_distributions.cylindrical.exponential_disk.F90

file merger_trees.branching_probability.Parkinson_Cole_Helly.F90

function node_component_disk_standard_density

function node_component_disk_standard_potential

function node_component_disk_standard_rotation_curve_gradient

subroutine node_component_disk_standard_radius_solver

file satellites.merging.virial_orbits.Wetzel2010.F90

file statistics.distributions.peak_background.F90

subroutine fspsreadfile

file structure_formation.cosmological_mass_variance.filtered_power_spectrum.F90

module gravitational_lensing

file structure_formation.halo_mass_function.Tinker2008.F90

subroutine simpleretabulate

subroutine spherical_collapse_dark_energy_critical_overdensity_tabulate

subroutine correlationfunctionfinalizeanalysis

function stellarvshalomassrelationleauthaud2012constructorinternal

file halo_model.conditional_mass_function.Behroozi2010.F90

subroutine halo_model_projected_correlation

subroutine interface_camb_transfer_function

file intergalactic_medium.filtering_mass.Gnedin2000.F90

file mass_distributions.cylindrical.Miyamoto_Nagai.F90

module math_exponentiation

file merger_trees.operators.particulate.F90

function node_component_disk_standard_enclosed_mass

function node_component_disk_standard_rotation_curve

function node_component_disk_standard_surface_density

file satellites.merging.virial_orbits.Jiang2014.F90

file star_formation.rate_surface_density.disks.Krumholz2009.F90

module stellar_populations_initial_mass_functions

file stellar_populations.spectra.dust_attenuation.tabulated.F90

file structure_formation.critical_overdensity.spherical_collapse_matter_lambda.F90

file structure_formation.halo_environment.normal.F90

file structure_formation.linear_growth.simple.F90

subroutine make_table

subroutine spherical_collapse_dark_energy_turnaround_radius_tabulate

subroutine spherical_collapse_dark_energy_virial_density_contrast_tabulate
subroutine make_table
subroutine restore_table
subroutine spherical_collape_matter_lambda_delta_virial_tabulate
subroutine spherical_collapse_matter_lambda_critical_overdensity_tabulate
subroutine spherical_collapse_matter_lambda_nonlinear_mapping
subroutine store_table
file structure_formation.transfer_function.CAMB.F90
file structure_formation.transfer_function.accelerator.F90
file structure_formation.transfer_function.file.F90
file structure_formation.virial_density_contrast.percolation.F90
file structure_formation.virial_density_contrast.spherical_collapse_matter_dark_energy.F90
file structure_formation.virial_density_contrast.spherical_collapse_matter_lambda.F90
program tests_spherical_collapse_nonlinear
program test_tables

type: `table`
Description: Basic table type.
Code lines: 13
Contained by: module tables

type: `table1d`
Description: Basic table type.
Code lines: 88
Contained by: module tables

function: `table1d_find_effective_x`
Description: Return the effective value of x to use in table interpolations.
Code lines: 31
Contained by: module tables
Modules used: galacticus_error

function: `table1d_integration_weights`
Description: Returns a set of weights for trapezoidal integration on the table between limits `x0` and `x1`.
Code lines: 27
Contained by: module tables
Modules used: galacticus_error

function: `table1d_is_monotonic`
Description: Return true if a 1D table is monotonic. Optionally allows specification of the direction, and whether or not equal elements are allowed for monotonicity.
Code lines: 14
Contained by: module tables
Modules used: array_utilities

function: `table1d_size`

Description: Return the size of a 1D table.
Code lines: 7
Contained by: module `tables`

function: `table1d_x`
Description: Return the `i`$^{\text{th}}$ x-value for a 1D table.
Code lines: 11
Contained by: module `tables`

function: `table1d_xs`
Description: Return the x-values for a 1D table.
Code lines: 8
Contained by: module `tables`

function: `table1d_y`
Description: Return the `i`$^{\text{th}}$ y-value for a 1D table.
Code lines: 14
Contained by: module `tables`

function: `table1d_ys`
Description: Return the y-values for a 1D table.
Code lines: 8
Contained by: module `tables`

type: `table1dgeneric`
Description: Table type supporting generic one dimensional tables.
Code lines: 29
Contained by: module `tables`

type: `table1dlinearcspline`
Description: Table type supporting one dimensional table with linear spacing in x and cubic spline interpolation.
Code lines: 30
Contained by: module `tables`

type: `table1dlinearlinear`
Description: Table type supporting one dimensional table with linear spacing in x.
Code lines: 26
Contained by: module `tables`

type: `table1dlinearmonotonecspline`
Description: Table type supporting one dimensional table with linear spacing in x and monotonic cubic spline interpolation.
Code lines: 26
Contained by: module `tables`

type: `table1dlogarithmiccspline`
Description: Table type supporting one dimensional table with logarithmic spacing in x and cubic spline interpolation.
Code lines: 11

Contained by: module tables

type: table1dlogarithmiclinear
Description: Table type supporting one dimensional table with logarithmic spacing in x.
Code lines: 12
Contained by: module tables

type: table1dlogarithmicmonotonecspline
Description: Table type supporting one dimensional table with logarithmic spacing in x and monotonic cubic spline interpolation.
Code lines: 11
Contained by: module tables

type: table1dnonuniformlinearlogarithmic
Description: Table type supporting one dimensional table with non-uniform x-axis and logarithmic in y.
Code lines: 10
Contained by: module tables

type: table2dlinlinlin
Description: Table type supporting generic two dimensional tables.
Code lines: 57
Contained by: module tables

type: table2dlogloglin
Description: Two-dimensional table type with logarithmic spacing in x and y dimensions, and linear interpolation in z.
Code lines: 105
Contained by: module tables

subroutine: table_1d_destroy
Description: Destroy a 1-D table.
Code lines: 9
Contained by: module tables
Modules used: memory_management

subroutine: table_1d_reverse
Description: Reverse a 1D table (i.e. swap x and y components). Optionally allows specification of which y table to swap with.
Code lines: 41
Contained by: module tables
Modules used: array_utilities galacticus_error

subroutine: table_2d_linlinlin_create
Description: Create a 2-D generic table.
Code lines: 24
Contained by: module tables
Modules used: galacticus_error memory_management

subroutine: table_2d_linlinlin_destroy
Description: Destroy a generic 2-D table.

Code lines: 13
Contained by: module tables
Modules used: memory_management numerical_interpolation

function: table_2d_linlinlin_interpolate
Description: Perform generic interpolation in a generic 2D table.
Code lines: 28
Contained by: module tables
Modules used: iso_c_binding numerical_interpolation

subroutine: table_2d_linlinlin_populate
Description: Populate a 2-D linear table.
Code lines: 21
Contained by: module tables
Modules used: galacticus_error

subroutine: table_2d_linlinlin_populate_single
Description: Populate a single element of a 2-D generic table.
Code lines: 22
Contained by: module tables
Modules used: galacticus_error

function: table_2d_linlinlin_xs
Description: Return the x-values for a 2D table.
Code lines: 8
Contained by: module tables

function: table_2d_linlinlin_ys
Description: Return the y-values for a 2D table.
Code lines: 8
Contained by: module tables

function: table_2d_linlinlin_zs
Description: Return the z-values for a 2D table.
Code lines: 8
Contained by: module tables

subroutine: table_2dlogloglin_create
Description: Create a 2-D log-log-linear table.
Code lines: 50
Contained by: module tables
Modules used: memory_management numerical_ranges

subroutine: table_2dlogloglin_destroy
Description: Destroy a 2D log-log-linear table.
Code lines: 10
Contained by: module tables
Modules used: memory_management

function: `table_2dlogloglin_interpolate`
Description: Perform linear interpolation in a logarithmic 1D table.
Code lines: 21
Contained by: module `tables`

function: `table_2dlogloglin_interpolate_gradient`
Description: Perform linear interpolation in a logarithmic 1D table.
Code lines: 30
Contained by: module `tables`
Modules used: `galacticus_error`

subroutine: `table_2dlogloglin_interpolation_factors`
Code lines: 39
Contained by: module `tables`

function: `table_2dlogloglin_is_initialized`
Description: Return true if a 2D log-log-linear table has been created.
Code lines: 7
Contained by: module `tables`

subroutine: `table_2dlogloglin_populate`
Description: Populate a 2-D log-log-linear table.
Code lines: 20
Contained by: module `tables`
Modules used: `galacticus_error`

subroutine: `table_2dlogloglin_populate_single`
Description: Populate a single element of a 2-D log-log-linear table.
Code lines: 22
Contained by: module `tables`
Modules used: `galacticus_error`

function: `table_2dlogloglin_size`
Description: Return the size of a 2D log-log-linear table.
Code lines: 17
Contained by: module `tables`
Modules used: `galacticus_error`

function: `table_2dlogloglin_x`
Description: Return the $\mathtt{i}^{\mathrm{th}}$ x-value for a 2D log-log table.
Code lines: 8
Contained by: module `tables`

function: `table_2dlogloglin_xs`
Description: Return the x-values for a 2D log-log table.
Code lines: 8
Contained by: module `tables`

function: `table_2dlogloglin_y`
Description: Return the `i`$^{\text{th}}$ y-value for a 2D log-log table.
Code lines: 8
Contained by: module `tables`

function: `table_2dlogloglin_ys`
Description: Return the y-values for a 2D log-log table.
Code lines: 8
Contained by: module `tables`

function: `table_2dlogloglin_z`
Description: Return the `(i,j)`$^{\text{th}}$ x-value for a 2D log-log table.
Code lines: 12
Contained by: module `tables`

function: `table_2dlogloglin_zs`
Description: Return the y-values for a 2D log-log table.
Code lines: 12
Contained by: module `tables`

subroutine: `table_generic_1d_create`
Description: Create a 1-D generic table.
Code lines: 35
Contained by: module `tables`
Modules used: `galacticus_error` `memory_management`

subroutine: `table_generic_1d_destroy`
Description: Destroy a generic 1-D table.
Code lines: 9
Contained by: module `tables`
Modules used: `numerical_interpolation`

function: `table_generic_1d_interpolate`
Description: Perform generic interpolation in a generic 1D table.
Code lines: 13
Contained by: module `tables`
Modules used: `numerical_interpolation`

function: `table_generic_1d_interpolate_gradient`
Description: Perform generic interpolation in a generic 1D table.
Code lines: 13
Contained by: module `tables`
Modules used: `numerical_interpolation`

subroutine: `table_generic_1d_populate`
Description: Populate a 1-D generic table.
Code lines: 20

Contained by: module tables
Modules used: galacticus_error

subroutine: table_generic_1d_populate_single
Description: Populate a single element of a 1-D generic table.
Code lines: 21
Contained by: module tables
Modules used: galacticus_error

subroutine: table_linear_1d_create
Description: Create a 1-D linear table.
Code lines: 34
Contained by: module tables
Modules used: galacticus_error memory_management numerical_ranges

function: table_linear_1d_interpolate
Description: Perform linear interpolation in a linear 1D table.
Code lines: 33
Contained by: module tables

function: table_linear_1d_interpolate_gradient
Description: Perform linear interpolation in a linear 1D table.
Code lines: 30
Contained by: module tables

subroutine: table_linear_1d_populate
Description: Populate a 1-D linear table.
Code lines: 24
Contained by: module tables
Modules used: galacticus_error

subroutine: table_linear_1d_populate_single
Description: Populate a single element of a 1-D linear table.
Code lines: 25
Contained by: module tables
Modules used: galacticus_error

subroutine: table_linear_cspline_1d_compute_spline
Description: Compute the interpolating spline factors for a 1-D linear spline.
Code lines: 41
Contained by: module tables

subroutine: table_linear_cspline_1d_create
Description: Create a 1-D linear table.
Code lines: 41
Contained by: module tables
Modules used: galacticus_error memory_management numerical_ranges

subroutine: `table_linear_cspline_1d_destroy`
Description: Destroy a linear cubic-sline 1-D table.
Code lines: 13
Contained by: module `tables`
Modules used: `memory_management`

function: `table_linear_cspline_1d_interpolate`
Description: Perform linear interpolation in a linear 1D table.
Code lines: 32
Contained by: module `tables`

function: `table_linear_cspline_1d_interpolate_gradient`
Description: Perform linear interpolation in a linear 1D table.
Code lines: 32
Contained by: module `tables`

subroutine: `table_linear_cspline_1d_populate`
Description: Populate a 1-D linear table.
Code lines: 27
Contained by: module `tables`
Modules used: `galacticus_error`

subroutine: `table_linear_cspline_1d_populate_single`
Description: Populate a single element of a 1-D linear table.
Code lines: 28
Contained by: module `tables`
Modules used: `galacticus_error`

function: `table_linear_cspline_integration_weights`
Description: Returns a set of weights for trapezoidal integration on the table between limits `x0` and `x1`.
Code lines: 13
Contained by: module `tables`
Modules used: `galacticus_error`

subroutine: `table_linear_monotone_cspline_1d_compute_spline`
Description: Compute the interpolating spline factors for a 1-D linear spline.
Code lines: 45
Contained by: module `tables`

subroutine: `table_linear_monotone_cspline_1d_create`
Description: Create a 1-D linear table.
Code lines: 36
Contained by: module `tables`
Modules used: `galacticus_error` `memory_management` `numerical_ranges`

subroutine: `table_linear_monotone_cspline_1d_destroy`

Description: Destroy a linear cubic-sline 1-D table.
Code lines: 11
Contained by: module tables
Modules used: memory_management

function: table_linear_monotone_cspline_1d_interpolate
Description: Perform linear interpolation in a linear 1D table.
Code lines: 32
Contained by: module tables

function: table_linear_monotone_cspline_1d_interpolate_gradient
Description: Perform linear interpolation in a linear 1D table.
Code lines: 32
Contained by: module tables

subroutine: table_linear_monotone_cspline_1d_populate
Description: Populate a 1-D linear table.
Code lines: 27
Contained by: module tables
Modules used: galacticus_error

subroutine: table_linear_monotone_cspline_1d_populate_single
Description: Populate a single element of a 1-D linear table.
Code lines: 28
Contained by: module tables
Modules used: galacticus_error

function: table_linear_monotone_cspline_integration_weights
Description: Returns a set of weights for trapezoidal integration on the table between limits x0 and x1.
Code lines: 13
Contained by: module tables
Modules used: galacticus_error

subroutine: table_logarithmic_1d_create
Description: Create a 1-D logarithmic table.
Code lines: 15
Contained by: module tables

function: table_logarithmic_1d_interpolate
Description: Perform linear interpolation in a logarithmic 1D table.
Code lines: 14
Contained by: module tables

function: table_logarithmic_1d_interpolate_gradient
Description: Perform linear interpolation in a logarithmic 1D table.
Code lines: 14
Contained by: module tables

subroutine: `table_logarithmic_1d_reverse`
Description: Reverse a 1D logarithmic-linear table (i.e. swap x and y components). Optionally allows specification of which y table to swap with.
Code lines: 35
Contained by: module `tables`
Modules used: `array_utilities` `galacticus_error`

function: `table_logarithmic_1d_x`
Description: Return the `i`$^{\text{th}}$ x-value for a logarithmic 1D table.
Code lines: 8
Contained by: module `tables`

function: `table_logarithmic_1d_xs`
Description: Return the x-values for a 1D table.
Code lines: 8
Contained by: module `tables`

subroutine: `table_logarithmic_cspline_1d_create`
Description: Create a 1-D logarithmic table.
Code lines: 18
Contained by: module `tables`

function: `table_logarithmic_cspline_1d_interpolate`
Description: Perform linear interpolation in a logarithmic 1D table.
Code lines: 14
Contained by: module `tables`

function: `table_logarithmic_cspline_1d_interpolate_gradient`
Description: Perform linear interpolation in a logarithmic 1D table.
Code lines: 14
Contained by: module `tables`

function: `table_logarithmic_cspline_1d_x`
Description: Return the `i`$^{\text{th}}$ x-value for a logarithmic 1D table.
Code lines: 16
Contained by: module `tables`

function: `table_logarithmic_cspline_1d_xs`
Description: Return the x-values for a 1D table.
Code lines: 8
Contained by: module `tables`

function: `table_logarithmic_integration_weights`
Description: Returns a set of weights for trapezoidal integration on the table between limits `x0` and `x1`.
Code lines: 76
Contained by: module `tables`
Modules used: `galacticus_error` `numerical_integration`

function: `factor0integrand`
Description: Integrand used to evaluate integration weights over logarithmically spaced tables
Code lines: 9
Contained by: function `table_logarithmic_integration_weights`

function: `factor1integrand`
Description: Integrand used to evaluate integration weights over logarithmically spaced tables
Code lines: 9
Contained by: function `table_logarithmic_integration_weights`

subroutine: `table_logarithmic_monotone_cspline_1d_create`
Description: Create a 1-D logarithmic table.
Code lines: 18
Contained by: module `tables`

function: `table_logarithmic_monotone_cspline_1d_interpolate`
Description: Perform linear interpolation in a logarithmic 1D table.
Code lines: 14
Contained by: module `tables`

function: `table_logarithmic_monotone_cspline_1d_interpolate_gradient`
Description: Perform linear interpolation in a logarithmic 1D table.
Code lines: 14
Contained by: module `tables`

function: `table_logarithmic_monotone_cspline_1d_x`
Description: Return the $\texttt{i}^{\text{th}}$ x-value for a logarithmic 1D table.
Code lines: 16
Contained by: module `tables`

function: `table_logarithmic_monotone_cspline_1d_xs`
Description: Return the x-values for a 1D table.
Code lines: 8
Contained by: module `tables`

function: `table_nonuniform_linear_logarithmic_1d_interpolate`
Description: Perform linear interpolation in a linear-logarithmic 1D table.
Code lines: 9
Contained by: module `tables`

function: `table_nonuniform_linear_logarithmic_1d_interpolate_gradient`
Description: Perform linear interpolation in a linear-logarithmic 1D table.
Code lines: 9
Contained by: module `tables`

subroutine: `table_nonuniform_linear_logarithmic_1d_populate`
Description: Populate a 1-D linear-logarihtmic table.
Code lines: 10

Contained by: module tables
Modules used: galacticus_error

subroutine: table_nonuniform_linear_logarithmic_1d_populate_single
Description: Populate a single element of a 1-D linear table.
Code lines: 11
Contained by: module tables
Modules used: galacticus_error

function: table_nonuniform_linear_logarithmic_1d_y
Description: Return the $\mathtt{i}^{\rm th}$ y-value for a 1D table.
Code lines: 14
Contained by: module tables

function: table_nonuniform_linear_logarithmic_1d_ys
Description: Return the y-values for a 1D table.
Code lines: 8
Contained by: module tables

function: table_nonuniform_linear_logarithmic_integration_weights
Description: Returns a set of weights for integration on a linear-logarithmic table between limits x0 and x1.
Code lines: 13
Contained by: module tables
Modules used: galacticus_error

file: objects.tables.labels.F90
Description: Contains a module which defines a labels for the table class.
Code lines: 38

module: table_labels
Description: Defines labels for the table class.
Code lines: 16
Contained by: file objects.tables.labels.F90
Used by:
function adafconstructorinternal
function ciefileelectrondensity
subroutine ciefilereadfile
function ciefilecoolingfunctiontemperaturelogslope
file dark_matter_-halos.spins.distributions.Bett2007.F90
subroutine correlationfunctionfinalizeanalysis
subroutine behroozi2010compute
subroutine halo_model_projected_-correlation
subroutine ciefilechemicaldensities
function ciefileelectrondensitytemperaturelogslope
function ciefilecoolingfunction
subroutine ciefilereadfile
function klypin2015constructorinternal
function disksizeinclinationconstructorinternal
function triaxialityconstructorinternal
subroutine interface_camb_transfer_-function

subroutine filereaddata	subroutine miyamotonagaimassenclosedtabulate
subroutine miyamotonagaisurfacedensitytabulate	subroutine exponentialdisktabulate
function sersicdensity	subroutine parkinsoncolehellysubresolutionhypergeometrictabulate
subroutine parkinsoncolehellyupperboundhypergeometrictabulate	subroutine sampleddistributionconstruct
subroutine particulatetabulateenergydistribution	function interpolate
function interpolate_derivative	module tables
function krumholz2009constructorinternal	function gordon2003constructorinternal
function prevotbouchetconstructorinternal	function prevotbouchetconstructorparameters
function wittgordon2003constructorinternal	function standardinterpolate
function barkana2001wdmgradientmass	function barkana2001wdmvalue
subroutine takahashi2011lensingdistributionconstruct	function rodriguezpuebla2016constructorinternal
function tinker2008constructorinternal	function tinker2008formdifferential
subroutine simpleretabulate	function cosmicemuvalue
subroutine spherical_collapse_matter_-lambda_nonlinear_mapping	subroutine filereadfile
program test_interpolation	subroutine xml_extrapolation_element_-decode

file: `objects.tensors.F90`

Description: Contains a module which defines the tensor symmetric structure used for describing symmetric tensors.
Code lines: 670

module: `tensors`

Description: Defines the tensor symmetric structure used for describing symmetric tensors.
Code lines: 646
Contained by: file objects.tensors.F90
Modules used: iso_varying_string, numerical_constants_astronomical
Used by: module galacticus_nodes; module node_component_satellite_-orbiting; function gnedin1999heatingrate; program test_tensors

interface: `assignment(=)`

Code lines: 3
Contained by: module tensors

interface: `max`

Code lines: 2
Contained by: module tensors

interface: `operator(*)`
Code lines: 2
Contained by: module tensors

type: `tensor`
Description: A generic tensor type.
Code lines: 2
Contained by: module tensors

function: `tensor_r2_d3_sym_add`
Description: Add two `tensorRank2Dimension3Symmetric` objects.
Code lines: 22
Contained by: module tensors

subroutine: `tensor_r2_d3_sym_assign_from`
Description: Assign a `tensorRank2Dimension3Symmetric` to a matrix.
Code lines: 16
Contained by: module tensors

subroutine: `tensor_r2_d3_sym_assign_to`
Description: Assign a matrix to a `tensorRank2Dimension3Symmetric` object.
Code lines: 9
Contained by: module tensors
Modules used: galacticus_error

subroutine: `tensor_r2_d3_sym_builder`
Description: Build a `tensorRank2Dimension3Symmetric` object from the given XML `tensorDefinition`.
Code lines: 34
Contained by: module tensors
Modules used: `fox_dom` galacticus_error

function: `tensor_r2_d3_sym_contract`
Description: Return the contraction (trace) of a `tensorRank2Dimension3Symmetric`.
Code lines: 7
Contained by: module tensors

subroutine: `tensor_r2_d3_sym_deserialize`
Description: Pack an array into a `tensorRank2Dimension3Symmetric` symmetric structure.
Code lines: 13
Contained by: module tensors

subroutine: `tensor_r2_d3_sym_destroy`
Description: Destroy a `tensorRank2Dimension3Symmetric` symmetric object.
Code lines: 8
Contained by: module tensors
Modules used: memory_management

function: `tensor_r2_d3_sym_double_contract`
Description: Find the double contraction of two `tensorRank2Dimension3Symmetric` objects, **A** : **B**.
Code lines: 7
Contained by: module tensors

subroutine: `tensor_r2_d3_sym_dump`
Description: Reset a `tensorRank2Dimension3Symmetric` symmetric object.
Code lines: 28
Contained by: module tensors
Modules used: galacticus_display iso_varying_string

subroutine: `tensor_r2_d3_sym_dump_raw`
Description: Dump a `tensorRank2Dimension3Symmetric` object to binary.
Code lines: 16
Contained by: module tensors
Modules used: galacticus_display iso_varying_string

subroutine: `tensor_r2_d3_sym_from_matrix`
Description: Construct a `tensorRank2Dimension3Symmetric` object from a matrix.
Code lines: 17
Contained by: module tensors
Modules used: galacticus_error

subroutine: `tensor_r2_d3_sym_increment`
Description: Increment a `tensorRank2Dimension3Symmetric` object.
Code lines: 13
Contained by: module tensors

function: `tensor_r2_d3_sym_is_zero`
Description: Test whether a `tensorRank2Dimension3Symmetric` object is zero.
Code lines: 8
Contained by: module tensors

function: `tensor_r2_d3_sym_matrix_equality`
Description: Return true if the supplied tensor and matrix are equal.
Code lines: 8
Contained by: module tensors

function: `tensor_r2_d3_sym_max`
Description: Return an element-by-element `max()` on two `tensorRank2Dimension3Symmetric` objects.
Code lines: 13
Contained by: module tensors

function: `tensor_r2_d3_sym_non_static_size_of`
Description: Return the size of any non-static components of the object.
Code lines: 10
Contained by: module tensors

Modules used: `iso_c_binding`

function: `tensor_r2_d3_sym_property_count`
Description: Return the number of properties required to track a rank 2, 3 dimensional, symmetric tensor. This is equal to 6.
Code lines: 7
Contained by: module tensors

subroutine: `tensor_r2_d3_sym_read_raw`
Description: Read a `tensorRank2Dimension3Symmetric` object from binary.
Code lines: 16
Contained by: module tensors
Modules used: galacticus_display iso_varying_string

subroutine: `tensor_r2_d3_sym_reset`
Description: Reset a `tensorRank2Dimension3Symmetric` object.
Code lines: 13
Contained by: module tensors

function: `tensor_r2_d3_sym_scalar_divide`
Description: Multiply a `tensorRank2Dimension3Symmetric` object by a scalar.
Code lines: 14
Contained by: module tensors

function: `tensor_r2_d3_sym_scalar_multiply`
Description: Multiply a `tensorRank2Dimension3Symmetric` object by a scalar.
Code lines: 14
Contained by: module tensors

function: `tensor_r2_d3_sym_scalar_multiply_switched`
Description: Multiply a scalar by a `tensorRank2Dimension3Symmetric` object.
Code lines: 9
Contained by: module tensors

subroutine: `tensor_r2_d3_sym_serialize`
Description: Pack a `tensorRank2Dimension3Symmetric` into an array.
Code lines: 14
Contained by: module tensors

subroutine: `tensor_r2_d3_sym_set_to_identity`
Description: Set a `tensorRank2Dimension3Symmetric` object to the identity matrix.
Code lines: 13
Contained by: module tensors

subroutine: `tensor_r2_d3_sym_set_to_unity`
Description: Set a `tensorRank2Dimension3Symmetric` object to unity.
Code lines: 13
Contained by: module tensors

function: `tensor_r2_d3_sym_subtract`
Description: Subtract two `tensorRank2Dimension3Symmetric` objects.
Code lines: 23
Contained by: module tensors

function: `tensor_r2_d3_sym_to_matrix`
Description: Construct a matrix from a `tensorRank2Dimension3Symmetric`.
Code lines: 16
Contained by: module tensors

type: `tensorrank2dimension3symmetric`
Description: A rank 2, three dimensional, symmetric tensor.
Code lines: 174
Contained by: module tensors

file: `output.times.F90`
Description: Contains a module which provides a class that implements output times for Galacticus.
Code lines: 71

module: `output_times`
Description: Provides a class that implements output times for Galacticus.
Code lines: 49
Contained by: file output.times.F90
Modules used: `iso_c_binding`
Used by:

subroutine galacticus_function_classes_destroy	function hivshalomassrelationpadmanabhan2017constructorin
function localgroupmassfunctionconstructorinternal	function localgroupmassfunctionconstructorparameters
function blackholebulgerelationconstructorinternal	function colordistributionsdssconstructorinternal
function concentrationdistributioncdmcococonstructorinternal	function concentrationvshalomasscdmludlow2016constructori
file galacticus.output.analyses.correlation_function.F90	file galacticus.output.analyses.distribution_operator.gravitational_lensing.F90
function galaxysizessdssconstructorinternal	function luminosityfunctionhalphaconstructorinternal
function massfunctionhiconstructorinternal	function massmetallicityandrews2013constructorinternal
function massmetallicityblanc2017constructorinternal	file galacticus.output.analyses.mean_function_1d.F90
function morphologicalfractiongamamoffett2016constructorinternal	file galacticus.output.analyses.property_operator.cosmology.angular_distance.F90

file galacticus.output.analyses.property_operator.cosmology.luminosity_distance.F90
file galacticus.output.analyses.scatter_function_1d.F90
function spindistributionbett2007constructorinternal
function output_analysis_output_weight_survey_volume
file galacticus.output.analyses.volume_function_1d.F90
function csmlgyvolumeoperate
subroutine galacticus_state_retrieve
subroutine galacticus_state_store
file merger_trees.construct.build.F90
file merger_trees.construct.read.F90
file merger_trees.evolve.timesteps.record_evolution.F90
file nodes.property_extractor.descendents.F90
file nodes.property_extractor.luminosity_emission_line.F90
function lmnstyemssnlineconstructorinternal
file nodes.property_extractor.luminosity_stellar.F90
file nodes.property_extractor.luminosity_stellar.dust.CharlotFall2000.F90
subroutine node_component_age_statistics_standard_rate_compute
module node_component_inter_output_standard
module node_component_merging_statistics_recent
subroutine stellar_luminosities_special_cases
file radiation.intergalactic_background.internal.F90
file star_formation.histories.in_situ.F90
file star_formation.histories.metallicity_split.F90
file stellar_populations.properties.noninstantaneous.F90
file tasks.evolve_forests.F90
file tasks.halo_mass_function.F90
file tasks.halo_spin_distribution.F90
file tasks.intergalactic_medium_state.F90
file tasks.power_spectrum.F90
file universe.operators.intergalactic_medium_state_evolve.F90

file: `output.times.list.F90`
Code lines: 230
Modules used: cosmology_functions

function: `listconstructorinternal`
Description: Constructor for the `list` output times class which takes a parameter set as input.
Code lines: 15
Contained by: file output.times.list.F90
Modules used: memory_management

function: `listconstructorparameters`
Description: Constructor for the `list` output times class which takes a parameter set as input.
Code lines: 53

Contained by: file output.times.list.F90
Modules used: array_utilities input_parameters memory_management sort

function: listcount
Description: Return the number of outputs.
Code lines: 8
Contained by: file output.times.list.F90

subroutine: listdestructor
Description: Destructor for the list output times class.
Code lines: 7
Contained by: file output.times.list.F90

function: listindex
Description: Returns the index of the output given the corresponding time.
Code lines: 19
Contained by: file output.times.list.F90
Modules used: arrays_search galacticus_error kind_numbers numerical_comparison

function: listredshift
Description: Returns the redshift of the output indexed by indexOutput.
Code lines: 12
Contained by: file output.times.list.F90

function: listtime
Description: Returns the time of the output indexed by iOutput.
Code lines: 12
Contained by: file output.times.list.F90

function: listtimenext
Description: Returns the time of the next output after currentTime.
Code lines: 23
Contained by: file output.times.list.F90
Modules used: arrays_search kind_numbers

function: listtimeprevious
Description: Returns the time of the previous output prior to timeCurrent.
Code lines: 17
Contained by: file output.times.list.F90
Modules used: arrays_search

interface: outputtimeslist
Description: Constructors for the list output times class.
Code lines: 4
Contained by: file output.times.list.F90

file: posterior_sampling.convergence.F90
Description: Contains a module which implements a class that provides convergence criteria for posterior sampling simulations.

Code lines: 63

module: `posterior_sampling_convergence`

Description: Implements a class that provides convergence criteria for posterior sampling simulations.
Code lines: 41
Contained by: file posterior_sampling.convergence.F90
Modules used: posterior_sampling_state
Used by:

subroutine galacticus_function_classes_destroy	subroutine galacticus_state_retrieve
subroutine galacticus_state_store	module models_likelihoods
function sedfitevaluate	function galaxypopulationevaluate
function galaxypopulationwillevaluate	function gaussianregressionevaluate
function gaussianregressionwillevaluate	function halomassfunctionevaluate
function massfunctionevaluate	function multivariatenormalevaluate
function multivariatenormalstochasticevaluate	function posteriorasprioreevaluate
function projectedcorrelationfunctionevaluate	function spindistributionevaluate
module posterior_sample_differential_proposal_size	module posterior_sampling_prop_size_temp_exp
file posterior_sampling.simulation.differential_evolution.F90	file posterior_sampling.simulation.particle_swarm.F90
module posterior_sampling_stopping_criteria	file posterior_sampling.stopping_criteria.correlation_length_count.F90
file posterior_sampling.stopping_criteria.step_count.F90	

file: `posterior_sampling.convergence.Gelman-Rubin.F90`

Description: Implementation of a posterior sampling convergence class which implements the Gelman-Rubin statistic.
Code lines: 422
Modules used: iso_varying_string

function: `gelmanrubinconstructorinternal`

Description: Constructor for "GelmanRubin" convergence class.
Code lines: 22
Contained by: file posterior_sampling.convergence.Gelman-Rubin.F90
Modules used:

galacticus_error	memory_management
mpi_utilities	

function: `gelmanrubinconstructorparameters`

Description: Constructor for the `gelmanRubin` posterior sampling convergence class which builds the object from a parameter set.
Code lines: 78
Contained by: file posterior_sampling.convergence.Gelman-Rubin.F90
Modules used:

input_parameters	iso_varying_string

function: `gelmanrubinconvergedatstep`

Description: Return the step at which the simulation converged.
Code lines: 7
Contained by: file posterior_sampling.convergence.Gelman-Rubin.F90

function: gelmanrubinconvergencemeasure
Description: Return the current maximum $\hat{R}$ convergence measure.
Code lines: 13
Contained by: file posterior_sampling.convergence.Gelman-Rubin.F90

function: gelmanrubinconvergencemeasuretarget
Description: Return the target $\hat{R}$ convergence measure.
Code lines: 7
Contained by: file posterior_sampling.convergence.Gelman-Rubin.F90

subroutine: gelmanrubindestructor
Description: Destroy a Gelman-Rubin convergence object.
Code lines: 9
Contained by: file posterior_sampling.convergence.Gelman-Rubin.F90
Modules used: mpi_utilities

function: gelmanrubinisconverged
Description: Return whether the simulation is converged.
Code lines: 165
Contained by: file posterior_sampling.convergence.Gelman-Rubin.F90
Modules used: fgsl, galacticus_display, iso_varying_string, memory_management, mpi_utilities, posterior_sampling_state, string_handling

subroutine: gelmanrubinlogreport
Description: Write a convergence report to the given fileUnit.
Code lines: 17
Contained by: file posterior_sampling.convergence.Gelman-Rubin.F90
Modules used: mpi_utilities

subroutine: gelmanrubinreset
Description: Reset the convergence object.
Code lines: 8
Contained by: file posterior_sampling.convergence.Gelman-Rubin.F90

function: gelmanrubinstateisoutlier
Description: Return true if the specified chain is deemed to be an outlier.
Code lines: 8
Contained by: file posterior_sampling.convergence.Gelman-Rubin.F90

interface: posteriorsampleconvergencegelmanrubin
Description: Constructors for the gelmanRubin posterior sampling convergence class.
Code lines: 4

Contained by: file posterior_sampling.convergence.Gelman-Rubin.F90

file: posterior_sampling.convergence.likelihood_threshold.F90
Description: Implementation of a posterior sampling convergence class which declares convergence once all likelihoods are above a threshold.
Code lines: 145

function: likelihoodthresholdconstructorinternal
Description: Internal constructor for the likelihoodThreshold posterior sampling convergence class.
Code lines: 11
Contained by: file posterior_sampling.convergence.likelihood_threshold.F90
Modules used: input_parameters

function: likelihoodthresholdconstructorparameters
Description: Constructor for the likelihoodThreshold posterior sampling convergence class which builds the object from a parameter set.
Code lines: 19
Contained by: file posterior_sampling.convergence.likelihood_threshold.F90
Modules used: input_parameters

function: likelihoodthresholdconvergedatstep
Description: Return the step at which the simulation converged.
Code lines: 7
Contained by: file posterior_sampling.convergence.likelihood_threshold.F90

function: likelihoodthresholdisconverged
Description: Returns true if the posterior sampling is converged (which it likelihoodThreshold is).
Code lines: 20
Contained by: file posterior_sampling.convergence.likelihood_threshold.F90
Modules used: mpi_utilities

subroutine: likelihoodthresholdlogreport
Description: Write a convergence report to the given fileUnit.
Code lines: 12
Contained by: file posterior_sampling.convergence.likelihood_threshold.F90

subroutine: likelihoodthresholdreset
Description: Reset the convergence object.
Code lines: 8
Contained by: file posterior_sampling.convergence.likelihood_threshold.F90

function: likelihoodthresholdstateisoutlier
Description: Return true if the specified chain is deemed to be an outlier. In this case, chains are likelihoodThreshold outliers.
Code lines: 9
Contained by: file posterior_sampling.convergence.likelihood_threshold.F90

interface: posteriorsampleconvergencelikelihoodthreshold
Description: Constructors for the likelihoodThreshold posterior sampling convergence class.
Code lines: 4

Contained by: file posterior_sampling.convergence.likelihood_threshold.F90

file: posterior_sampling.convergence.never.F90
Description: Implementation of a posterior sampling convergence class which never converges.
Code lines: 106

function: neverconstructorparameters
Description: Constructor for the never merger tree halo mass function sampling class which builds the object from a parameter set.
Code lines: 10
Contained by: file posterior_sampling.convergence.never.F90
Modules used: input_parameters

function: neverconvergedatstep
Description: Return the step at which the simulation converged.
Code lines: 8
Contained by: file posterior_sampling.convergence.never.F90

function: neverisconverged
Description: Returns true if the posterior sampling is converged (which it never is).
Code lines: 10
Contained by: file posterior_sampling.convergence.never.F90

subroutine: neverlogreport
Description: Write a convergence report to the given fileUnit.
Code lines: 9
Contained by: file posterior_sampling.convergence.never.F90

subroutine: neverreset
Description: Reset the convergence object.
Code lines: 7
Contained by: file posterior_sampling.convergence.never.F90

function: neverstateisoutlier
Description: Return true if the specified chain is deemed to be an outlier. In this case, chains are never outliers.
Code lines: 9
Contained by: file posterior_sampling.convergence.never.F90

interface: posteriorsampleconvergencenever
Description: Constructors for the never posterior sampling convergence class.
Code lines: 3
Contained by: file posterior_sampling.convergence.never.F90

file: posterior_sampling.differential_proposal_size.F90
Description: Contains a module which implements algorithms for the proposal size in differential evolution algorithms.
Code lines: 41

module: posterior_sample_differential_proposal_size

Description: Implements algorithms for the proposal size in differential evolution algorithms.
Code lines: 19
Contained by: file posterior_sampling.differential_proposal_size.F90
Modules used: posterior_sampling_convergence, posterior_sampling_state
Used by: subroutine galacticus_function_classes_destroy, subroutine galacticus_state_retrieve, subroutine galacticus_state_store, file posterior_sampling.simulation.differential_evolution.F90

file: posterior_sampling.differential_proposal_size.adaptive.F90
Description: Implementation of a posterior sampling differential evolution proposal size class in which the proposal size is adaptive.
Code lines: 165

function: adaptiveconstructorinternal
Description: Constructor for the "adaptive" differential evolution proposal size class.
Code lines: 11
Contained by: file posterior_sampling.differential_proposal_size.adaptive.F90

function: adaptiveconstructorparameters
Description: Constructor for the adaptive posterior sampling differential evolution random jump class which builds the object from a parameter set.
Code lines: 62
Contained by: file posterior_sampling.differential_proposal_size.adaptive.F90
Modules used: input_parameters

function: adaptivegamma
Description: Return the proposal size.
Code lines: 44
Contained by: file posterior_sampling.differential_proposal_size.adaptive.F90
Modules used: galacticus_display, iso_varying_string, mpi_utilities, string_handling

interface: posteriorsampledffrntlevltnproposalsizeadaptive
Description: Constructors for the adaptive posterior sampling differential evolution random jump class.
Code lines: 4
Contained by: file posterior_sampling.differential_proposal_size.adaptive.F90

file: posterior_sampling.differential_proposal_size.fixed.F90
Description: Implementation of a posterior sampling differential evolution proposal size class in which the proposal size is fixed.
Code lines: 82

function: fixedconstructorinternal
Description: Internal constructor for the fixed posterior sampling differential evolution random jump class.
Code lines: 8
Contained by: file posterior_sampling.differential_proposal_size.fixed.F90

function: `fixedconstructorparameters`
Description: Constructor for the `fixed` posterior sampling differential evolution random jump class which builds the object from a parameter set.
Code lines: 19
Contained by: file posterior_sampling.differential_proposal_size.fixed.F90
Modules used: input_parameters

function: `fixedgamma`
Description: Return the current state.
Code lines: 10
Contained by: file posterior_sampling.differential_proposal_size.fixed.F90

interface: `posteriorsampledffrntlevltnproposalsizefixed`
Description: Constructors for the `fixed` posterior sampling differential evolution random jump class.
Code lines: 4
Contained by: file posterior_sampling.differential_proposal_size.fixed.F90

file: `posterior_sampling.differential_proposal_size.temperature_exponent.F90`
Description: Contains a module which implements algorithms for the temperature exponent of proposal size in tempered differential evolution algorithms.
Code lines: 45

module: `posterior_sampling_prop_size_temp_exp`
Description: Implements algorithms for the temperature exponent of proposal size in tempered differential evolution algorithms.
Code lines: 22
Contained by: file posterior_sampling.differential_proposal_size.temperature_exponent.F90
Modules used: posterior_sampling_convergence, posterior_sampling_state
Used by: subroutine galacticus_function_classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; file posterior_sampling.simulation.differential_evolution.F90

file: `posterior_sampling.differential_proposal_size.temperature_exponent.adaptive.F90`
Description: Implementation of a posterior sampling differential evolution proposal size temperature exponent class in which the exponent is adaptive.
Code lines: 192

function: `adaptiveconstructorinternal`
Description: Constructor for the "adaptive" differential evolution proposal size class.
Code lines: 11
Contained by: file posterior_sampling.differential_proposal_size.temperature_exponent.adaptive.F90

function: `adaptiveconstructorparameters`
Description: Constructor for the `adaptive` posterior sampling differential evolution random jump class which builds the object from a parameter set.
Code lines: 62
Contained by: file posterior_sampling.differential_proposal_size.temperature_exponent.adaptive.F90

Modules used: input_parameters

function: adaptiveexponent
Description: Return the adaptive differential evolution proposal size temperature exponent.
Code lines: 70
Contained by: file posterior_sampling.differential_proposal_size.temperature_exponent.adaptive.F90
Modules used: galacticus_display, iso_varying_string, mpi_utilities, string_handling

interface: posteriorsampledffrntlevltnprpslsztmpexpadaptive
Description: Constructors for the adaptive posterior sampling differential evolution random jump class.
Code lines: 4
Contained by: file posterior_sampling.differential_proposal_size.temperature_exponent.adaptive.F90

file: posterior_sampling.differential_proposal_size.temperature_exponent.fixed.F90
Description: Implementation of a posterior sampling differential evolution proposal size temperature exponent class in which the exponent is fixed.
Code lines: 87

function: fixedconstructorinternal
Description: Internal constructor for the fixed posterior sampling differential evolution proposal size temperature exponent class.
Code lines: 9
Contained by: file posterior_sampling.differential_proposal_size.temperature_exponent.fixed.F90

function: fixedconstructorparameters
Description: Constructor for the fixed posterior sampling differential evolution random jump class which builds the object from a parameter set.
Code lines: 19
Contained by: file posterior_sampling.differential_proposal_size.temperature_exponent.fixed.F90
Modules used: input_parameters

function: fixedexponent
Description: Return the fixed differential evolution proposal size temperature exponent.
Code lines: 12
Contained by: file posterior_sampling.differential_proposal_size.temperature_exponent.fixed.F90

interface: posteriorsampledffrntlevltnprpslsztmpexpfixed
Description: Constructors for the fixed posterior sampling differential evolution random jump class.
Code lines: 4
Contained by: file posterior_sampling.differential_proposal_size.temperature_exponent.fixed.F90

file: posterior_sampling.differential_random_jump.F90
Description: Contains a module which implements class for the random jump component in differential evolution algorithms.
Code lines: 41

module: posterior_sample_differential_random_jump
Description: Implements a class for the random jump component in differential evolution algorithms.

Code lines: 19
Contained by: file posterior_sampling.differential_random_jump.F90
Modules used: model_parameters, posterior_sampling_state
Used by: subroutine galacticus_function_classes_destroy, subroutine galacticus_state_retrieve, subroutine galacticus_state_store, file posterior_sampling.simulation.differential_evolution.F90

file: posterior_sampling.differential_random_jump.adaptive.F90
Description: Implementation of a posterior sampling differential evolution random jump class in which the jump is drawn from an adaptive distribution which scales with the range spanned by the sample states.
Code lines: 73

function: adaptiveconstructorparameters
Description: Constructor for the adaptive posterior sampling differential evolution random jump class which builds the object from a parameter set.
Code lines: 11
Contained by: file posterior_sampling.differential_random_jump.adaptive.F90
Modules used: input_parameters

function: adaptivesample
Description: Sample from the random jump distribution.
Code lines: 18
Contained by: file posterior_sampling.differential_random_jump.adaptive.F90
Modules used: mpi_utilities

interface: posteriorsampledffrntlevltnrandomjumpadaptive
Description: Constructors for the adaptive posterior sampling differential evolution random jump class.
Code lines: 3
Contained by: file posterior_sampling.differential_random_jump.adaptive.F90

file: posterior_sampling.differential_random_jump.simple.F90
Description: Implementation of a posterior sampling differential evolution random jump class in which the jump is drawn from a fixed distribution.
Code lines: 69

interface: posteriorsampledffrntlevltnrandomjumpsimple
Description: Constructors for the simple posterior sampling differential evolution random jump class.
Code lines: 3
Contained by: file posterior_sampling.differential_random_jump.simple.F90

function: simpleconstructorparameters
Description: Constructor for the simple posterior sampling differential evolution random jump class which builds the object from a parameter set.
Code lines: 11
Contained by: file posterior_sampling.differential_random_jump.simple.F90
Modules used: input_parameters

function: `simplesample`
Description: Sample from the random jump distribution.
Code lines: 15
Contained by: file posterior_sampling.differential_random_jump.simple.F90

file: `posterior_sampling.simulation.F90`
Description: Contains a module which implements a class of posterior sampling simulators.
Code lines: 38

module: `posterior_sampling_simulation`
Description: Implements a class of posterior sampling simulators.
Code lines: 16
Contained by: file posterior_sampling.simulation.F90
Used by: subroutine galacticus_function_classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; file tasks.posterior_sample.F90

file: `posterior_sampling.simulation.annealed_differential_evolution.F90`
Description: Implementation of a posterior sampling simulation class which implements an annealed differential evolution algorithm.
Code lines: 200

function: `annealeddifferentialevolutionacceptproposal`
Description: Return whether or not to accept a proposal.
Code lines: 15
Contained by: file posterior_sampling.simulation.annealed_differential_evolution.F90
Modules used: mpi_utilities; pseudo_random

function: `annealeddifferentialevolutionconstructorinternal`
Description: Internal constructor for the "annealedDifferentialEvolution" simulation class.
Code lines: 20
Contained by: file posterior_sampling.simulation.annealed_differential_evolution.F90

function: `annealeddifferentialevolutionconstructorparameters`
Description: Constructor for the `annealedDifferentialEvolution` posterior sampling simulation class which builds the object from a parameter set.
Code lines: 29
Contained by: file posterior_sampling.simulation.annealed_differential_evolution.F90
Modules used: input_parameters

subroutine: `annealeddifferentialevolutioninitialize`
Description: Finished initialization of annealed differential evolution simulation objects during construction.
Code lines: 19
Contained by: file posterior_sampling.simulation.annealed_differential_evolution.F90

function: `annealeddifferentialevolutiontemperature`
Description: Return the temperature.
Code lines: 7

Contained by: file posterior_sampling.simulation.annealed_differential_evolution.F90

subroutine: annealeddifferentialevolutionupdate

Description: Update the differential evolution simulator state.
Code lines: 39
Contained by: file posterior_sampling.simulation.annealed_differential_evolution.F90
Modules used: galacticus_display, iso_varying_string, mpi_utilities, string_handling

interface: posteriorsamplesimulationannealeddffrntlevltn

Description: Constructors for the annealedDifferentialEvolution posterior sampling convergence class.
Code lines: 4
Contained by: file posterior_sampling.simulation.annealed_differential_evolution.F90

file: posterior_sampling.simulation.differential_evolution.F90

Description: Implementation of a posterior sampling simulation class which implements the differential evolution algorithm.
Code lines: 768
Modules used: model_parameters, models_likelihoods, posterior_sample_differential_-proposal_size, posterior_sample_differential_-random_jump, posterior_sampling_convergence, posterior_sampling_prop_size_temp_exp, posterior_sampling_state, posterior_sampling_state_initialize, posterior_sampling_stopping_criteria

function: differentialevolutionacceptproposal

Description: Return whether or not to accept a proposal.
Code lines: 15
Contained by: file posterior_sampling.simulation.differential_evolution.F90
Modules used: mpi_utilities, pseudo_random

function: differentialevolutionchainselect

Description: Select a chain at random, optionally excluding blocked chains.
Code lines: 18
Contained by: file posterior_sampling.simulation.differential_evolution.F90
Modules used: mpi_utilities, pseudo_random

function: differentialevolutionconstructorinternal

Description: Internal constructor for the “differentialEvolution” simulation class.
Code lines: 34
Contained by: file posterior_sampling.simulation.differential_evolution.F90

function: differentialevolutionconstructorparameters

Description: Constructor for the differentialEvolution posterior sampling simulation class which builds the object from a parameter set.
Code lines: 174
Contained by: file posterior_sampling.simulation.differential_evolution.F90
Modules used: galacticus_display, galacticus_error, input_parameters, mpi_utilities

string_handling

subroutine: differentialevolutiondestructor
Description: Destroy a differential evolution simulation object.
Code lines: 22
Contained by: file posterior_sampling.simulation.differential_evolution.F90

function: differentialevolutionlogging
Description: Specifies whether or not the current state should be logged to file during differential evolution.
Code lines: 8
Contained by: file posterior_sampling.simulation.differential_evolution.F90

subroutine: differentialevolutionposterior
Description: Return the log of the posterior for the current state.
Code lines: 157
Contained by: file posterior_sampling.simulation.differential_evolution.F90
Modules used: galacticus_display, galacticus_error, iso_c_binding, kind_numbers, mpi_utilities, sort

subroutine: differentialevolutionsimulate
Description: Perform a differential evolution simulation.
Code lines: 169
Contained by: file posterior_sampling.simulation.differential_evolution.F90
Modules used: file_utilities, galacticus_display, galacticus_error, kind_numbers, models_likelihoods_constants, mpi_utilities, pseudo_random, string_handling, system_command

function: differentialevolutionstepsize
Description: Return the step size parameter, γ, for a differential evolution step.
Code lines: 13
Contained by: file posterior_sampling.simulation.differential_evolution.F90

function: differentialevolutiontemperature
Description: Return the temperature.
Code lines: 8
Contained by: file posterior_sampling.simulation.differential_evolution.F90

subroutine: differentialevolutionupdate
Description: Update the differential evolution simulator state.
Code lines: 15
Contained by: file posterior_sampling.simulation.differential_evolution.F90
Modules used: mpi_utilities

interface: posteriorsamplesimulationdifferentialevolution
Description: Constructors for the differentialEvolution posterior sampling convergence class.

Code lines: 4
Contained by: file posterior_sampling.simulation.differential_evolution.F90

file: posterior_sampling.simulation.particle_swarm.F90
Description: Implementation of a posterior sampling simulation class which implements the differential evolution algorithm.
Code lines: 666
Modules used: model_parameters, models_likelihoods, posterior_sampling_convergence, posterior_sampling_state, posterior_sampling_state_initialize, posterior_sampling_stopping_criteria

function: particleswarmconstructorinternal
Description: Internal constructor for the "particleSwarm" simulation class.
Code lines: 31
Contained by: file posterior_sampling.simulation.particle_swarm.F90

function: particleswarmconstructorparameters
Description: Constructor for the particleSwarm posterior sampling simulation class which builds the object from a parameter set.
Code lines: 169
Contained by: file posterior_sampling.simulation.particle_swarm.F90
Modules used: galacticus_display, galacticus_error, input_parameters, mpi_utilities, string_handling

subroutine: particleswarmdestructor
Description: Destroy a differential evolution simulation object.
Code lines: 18
Contained by: file posterior_sampling.simulation.particle_swarm.F90

subroutine: particleswarmposterior
Description: Return the log of the posterior for the current state.
Code lines: 146
Contained by: file posterior_sampling.simulation.particle_swarm.F90
Modules used: galacticus_display, galacticus_error, iso_c_binding, kind_numbers, models_likelihoods_constants, mpi_utilities, sort

subroutine: particleswarmsimulate
Description: Perform a particle swarm simulation.
Code lines: 226
Contained by: file posterior_sampling.simulation.particle_swarm.F90
Modules used: error_functions, file_utilities, galacticus_display, galacticus_error, kind_numbers, models_likelihoods_constants, mpi_utilities, pseudo_random, string_handling, system_command

interface: `posteriorsamplesimulationparticleswarm`
Description: Constructors for the `particleSwarm` posterior sampling convergence class.
Code lines: 4
Contained by: file posterior_sampling.simulation.particle_swarm.F90

file: `posterior_sampling.simulation.stochastic_differential_evolution.F90`
Description: Implementation of a posterior sampling simulation class which implements a stochastic differential evolution algorithm.
Code lines: 138

interface: `posteriorsamplesimulationstochasticdffrntlevltn`
Description: Constructors for the `stochasticDifferentialEvolution` posterior sampling convergence class.
Code lines: 4
Contained by: file posterior_sampling.simulation.stochastic_differential_evolution.F90

function: `stochasticdifferentialevolutionacceptproposal`
Description: Return whether or not to accept a proposal.
Code lines: 33
Contained by: file posterior_sampling.simulation.stochastic_differential_evolution.F90
Modules used: galacticus_error mpi_utilities pseudo_random

function: `stochasticdifferentialevolutionconstructorinternal`
Description: Internal constructor for the "stochasticDifferentialEvolution" simulation class.
Code lines: 20
Contained by: file posterior_sampling.simulation.stochastic_differential_evolution.F90

function: `stochasticdifferentialevolutionconstructorparameters`
Description: Constructor for the `stochasticDifferentialEvolution` posterior sampling simulation class which builds the object from a parameter set.
Code lines: 20
Contained by: file posterior_sampling.simulation.stochastic_differential_evolution.F90
Modules used: input_parameters

subroutine: `stochasticdifferentialevolutioninitialize`
Description: Finished initialization of stochastic differential evolution simulation objects during construction.
Code lines: 8
Contained by: file posterior_sampling.simulation.stochastic_differential_evolution.F90

file: `posterior_sampling.simulation.tempered_differential_evolution.F90`
Description: Implementation of a posterior sampling simulation class which implements a tempered differential evolution algorithm.
Code lines: 327

interface: `posteriorsamplesimulationtempereddffrntlevltn`
Description: Constructors for the `temperedDifferentialEvolution` posterior sampling convergence class.
Code lines: 4

Contained by: file posterior_sampling.simulation.tempered_differential_evolution.F90

function: `temperedifferentialevolutionacceptproposal`
Description: Return whether or not to accept a proposal.
Code lines: 15
Contained by: file posterior_sampling.simulation.tempered_differential_evolution.F90
Modules used: mpi_utilities pseudo_random

function: `temperedifferentialevolutionconstructorinternal`
Description: Internal constructor for the "temperedDifferentialEvolution" simulation class.
Code lines: 21
Contained by: file posterior_sampling.simulation.tempered_differential_evolution.F90

function: `temperedifferentialevolutionconstructorparameters`
Description: Constructor for the `temperedDifferentialEvolution` posterior sampling simulation class which builds the object from a parameter set.
Code lines: 48
Contained by: file posterior_sampling.simulation.tempered_differential_evolution.F90
Modules used: input_parameters

subroutine: `temperedifferentialevolutiondestructor`
Description: Destroy a tempered differential evolution simulation object.
Code lines: 7
Contained by: file posterior_sampling.simulation.tempered_differential_evolution.F90

subroutine: `temperedifferentialevolutioninitialize`
Description: Finished initialization of tempered differential evolution simulation objects during construction.
Code lines: 25
Contained by: file posterior_sampling.simulation.tempered_differential_evolution.F90

function: `temperedifferentialevolutionlevel`
Description: Return the actual tempering level.
Code lines: 8
Contained by: file posterior_sampling.simulation.tempered_differential_evolution.F90

function: `temperedifferentialevolutionlogging`
Description: Specifies whether or not the current state should be logged to file during differential evolution.
Code lines: 7
Contained by: file posterior_sampling.simulation.tempered_differential_evolution.F90

function: `temperedifferentialevolutionstepsize`
Description: Return the step size parameter, γ, for a differential evolution step.
Code lines: 14
Contained by: file posterior_sampling.simulation.tempered_differential_evolution.F90

function: `temperedifferentialevolutiontemperature`
Description: Return the temperature.

Code lines: 11
Contained by: file posterior_sampling.simulation.tempered_differential_evolution.F90

subroutine: tempereddifferentialevolutionupdate
Description: Update the differential evolution simulator state.
Code lines: 85
Contained by: file posterior_sampling.simulation.tempered_differential_evolution.F90
Modules used: galacticus_display, iso_varying_string, mpi_utilities, string_handling

file: posterior_sampling.state.F90
Description: Contains a module which implements a class that maintains during posterior sampling.
Code lines: 117

module: posterior_sampling_state
Description: Implements a class that maintains during posterior sampling.
Code lines: 95
Contained by: file posterior_sampling.state.F90
Used by: subroutine galacticus_function_classes_destroy, subroutine galacticus_state_retrieve, subroutine galacticus_state_store, module models_likelihoods, function sedfitevaluate, function galaxypopulationevaluate, function galaxypopulationwillevaluate, function gaussianregressionevaluate, function gaussianregressionwillevaluate, function halomassfunctionevaluate, function massfunctionevaluate, function multivariatenormalevaluate, function multivariatenormalstochasticevaluate, function posteriorasprioreevaluate, function projectedcorrelationfunctionevaluate, function spindistributionevaluate, function modelparameterlistlogprior, module posterior_sampling_convergence, function gelmanrubinisconverged, module posterior_sample_differential_proposal_size, module posterior_sampling_prop_size_temp_exp, module posterior_sample_differential_random_jump, file posterior_sampling.simulation.differential_evolution.F90, file posterior_sampling.simulation.particle_swarm.F90, module posterior_sampling_state_initialize, subroutine resumeinitialize, module posterior_sampling_stopping_criteria

file: posterior_sampling.state.correlation.F90
Description: Implementation of a posterior sampling state class which computes correlation lengths.
Code lines: 269

function: correlationconstructorinternal
Description: Constructor for the correlation posterior sampling state class which builds the object from a parameter set.

Code lines: 17
Contained by: file posterior_sampling.state.correlation.F90
Modules used: mpi_utilities

function: correlationconstructorparameters
Description: Constructor for the correlation posterior sampling state class which builds the object from a parameter set.
Code lines: 20
Contained by: file posterior_sampling.state.correlation.F90
Modules used: input_parameters

function: correlationcorrelationlength
Description: Return the correlation length.
Code lines: 8
Contained by: file posterior_sampling.state.correlation.F90

subroutine: correlationcorrelationlengthcompute
Description: Compute correlation lengths.
Code lines: 39
Contained by: file posterior_sampling.state.correlation.F90
Modules used: galacticus_display iso_varying_string mpi_utilities string_handling

subroutine: correlationparametercountset
Description: Set the number of parameters in this state.
Code lines: 16
Contained by: file posterior_sampling.state.correlation.F90

function: correlationpostconvergencecorrelationcount
Description: Return the number of post-convergence correlation lengths that have accrued.
Code lines: 7
Contained by: file posterior_sampling.state.correlation.F90

subroutine: correlationreset
Description: Reset the state object.
Code lines: 10
Contained by: file posterior_sampling.state.correlation.F90

subroutine: correlationrestore
Description: Restore the state object from file.
Code lines: 23
Contained by: file posterior_sampling.state.correlation.F90
Modules used: iso_varying_string mpi_utilities

subroutine: correlationupdate
Description: Update the current state.
Code lines: 43
Contained by: file posterior_sampling.state.correlation.F90

interface: `posteriorsamplestatecorrelation`
Description: Constructors for the `correlation` posterior sampling state class.
Code lines: 4
Contained by: file `posterior_sampling.state.correlation.F90`

file: `posterior_sampling.state.history.F90`
Description: Implementation of a posterior sampling state class which stores history.
Code lines: 164

function: `historyconstructorinternal`
Description: Constructor for the `history` posterior sampling state class which builds the object from a parameter set.
Code lines: 14
Contained by: file `posterior_sampling.state.history.F90`
Modules used: `mpi_utilities`

function: `historyconstructorparameters`
Description: Constructor for the `history` posterior sampling state class which builds the object from a parameter set.
Code lines: 20
Contained by: file `posterior_sampling.state.history.F90`
Modules used: `input_parameters`

function: `historymean`
Description: Return the mean over state history.
Code lines: 9
Contained by: file `posterior_sampling.state.history.F90`
Modules used: `galacticus_error`

subroutine: `historyparametercountset`
Description: Set the number of parameters in this state.
Code lines: 11
Contained by: file `posterior_sampling.state.history.F90`

subroutine: `historyreset`
Description: Reset the state object.
Code lines: 9
Contained by: file `posterior_sampling.state.history.F90`

subroutine: `historyrestore`
Description: Restore the state object from file.
Code lines: 17
Contained by: file `posterior_sampling.state.history.F90`
Modules used: `iso_varying_string` `mpi_utilities`

subroutine: `historyupdate`
Description: Update the current state.
Code lines: 15

Contained by: file posterior_sampling.state.history.F90

function: historyvariance
Description: Return the mean over state history.
Code lines: 9
Contained by: file posterior_sampling.state.history.F90
Modules used: galacticus_error

interface: posteriorsamplestatehistory
Description: Constructors for the history posterior sampling state class.
Code lines: 4
Contained by: file posterior_sampling.state.history.F90

file: posterior_sampling.state.initialize.F90
Description: Contains a module which implements a class for posterior sampling state initialization.
Code lines: 45

module: posterior_sampling_state_initialize
Description: Implements a class for posterior sampling state initialization.
Code lines: 23
Contained by: file posterior_sampling.state.initialize.F90
Modules used: model_parameters, models_likelihoods, posterior_sampling_state
Used by: subroutine galacticus_function_classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; file posterior_sampling.simulation.differential_evolution.F90; file posterior_sampling.simulation.particle_swarm.F90

file: posterior_sampling.state.initialize.latin_hypercube.F90
Description: Implementation of a posterior sampling convergence class which latinHypercube converges.
Code lines: 141

function: latinhypercubeconstructorinternal
Description: Constructor for the latinHypercube posterior sampling state initialization class.
Code lines: 8
Contained by: file posterior_sampling.state.initialize.latin_hypercube.F90

function: latinhypercubeconstructorparameters
Description: Constructor for the latinHypercube posterior sampling state initialization class.
Code lines: 19
Contained by: file posterior_sampling.state.initialize.latin_hypercube.F90
Modules used: input_parameters

subroutine: latinhypercubeinitialize
Description: Initialize simulation state by drawing at random from the parameter priors.

Code lines: 69
Contained by: file posterior_sampling.state.initialize.latin_hypercube.F90
Modules used: iso_c_binding, models_likelihoods_constants, mpi_utilities, pseudo_random, sort

interface: posteriorsamplestateinitializelatinhypercube
Description: Constructors for the latinHypercube posterior sampling state initialization class.
Code lines: 4
Contained by: file posterior_sampling.state.initialize.latin_hypercube.F90

file: posterior_sampling.state.initialize.prior_random.F90
Description: Implementation of a posterior sampling convergence class which priorRandom converges.
Code lines: 75

interface: posteriorsamplestateinitializepriorrandom
Description: Constructors for the priorRandom posterior sampling state initialization class.
Code lines: 3
Contained by: file posterior_sampling.state.initialize.prior_random.F90

function: priorrandomconstructorparameters
Description: Constructor for the priorRandom posterior sampling state initialization class.
Code lines: 10
Contained by: file posterior_sampling.state.initialize.prior_random.F90
Modules used: input_parameters

subroutine: priorrandominitialize
Description: Initialize simulation state by drawing at random from the parameter priors.
Code lines: 24
Contained by: file posterior_sampling.state.initialize.prior_random.F90
Modules used: models_likelihoods_constants

file: posterior_sampling.state.initialize.resume.F90
Description: Implementation of a posterior sampling convergence class which resume converges.
Code lines: 151
Modules used: iso_varying_string

interface: posteriorsamplestateinitializeresume
Description: Constructors for the resume posterior sampling state initialization class.
Code lines: 4
Contained by: file posterior_sampling.state.initialize.resume.F90

function: resumeconstructorinternal
Description: Constructor for the resume posterior sampling state initialization class.
Code lines: 9
Contained by: file posterior_sampling.state.initialize.resume.F90

function: resumeconstructorparameters
Description: Constructor for the resume posterior sampling state initialization class.

Code lines: 26
Contained by: file posterior_sampling.state.initialize.resume.F90
Modules used: input_parameters

subroutine: resumeinitialize
Description: Initialize simulation state by drawing at random from the parameter priors.
Code lines: 68
Contained by: file posterior_sampling.state.initialize.resume.F90
Modules used: galacticus_display, galacticus_error, models_likelihoods_constants, mpi_utilities, posterior_sampling_state, string_handling

file: posterior_sampling.state.initialize.switched.F90
Description: Implementation of a posterior sampling convergence class which switches between two other options.
Code lines: 197

interface: posteriorsamplestateinitializeswitched
Description: Constructors for the switched posterior sampling state initialization class.
Code lines: 4
Contained by: file posterior_sampling.state.initialize.switched.F90

function: switchedconstructorinternal
Description: Constructor for the switched posterior sampling state initialization class.
Code lines: 9
Contained by: file posterior_sampling.state.initialize.switched.F90

function: switchedconstructorparameters
Description: Constructor for the switched posterior sampling state initialization class.
Code lines: 32
Contained by: file posterior_sampling.state.initialize.switched.F90
Modules used: input_parameters

subroutine: switcheddestructor
Description: Destructor for the switched posterior sampling state initialization class.
Code lines: 8
Contained by: file posterior_sampling.state.initialize.switched.F90

subroutine: switchedinitialize
Description: Initialize simulation state by drawing at random from the parameter priors.
Code lines: 99
Contained by: file posterior_sampling.state.initialize.switched.F90
Modules used: galacticus_error, models_likelihoods_constants

file: posterior_sampling.state.simple.F90
Description: Implementation of a simple posterior sampling state class.
Code lines: 207

interface: posteriorsamplestatesimple
Description: Constructors for the simple posterior sampling state class.

Code lines: 4
Contained by: file posterior_sampling.state.simple.F90

function: simpleacceptancerate
Description: Return the acceptance rate for the state.
Code lines: 13
Contained by: file posterior_sampling.state.simple.F90

function: simpleconstructorinternal
Description: Constructor for the simple posterior sampling state class which builds the object from a parameter set.
Code lines: 14
Contained by: file posterior_sampling.state.simple.F90
Modules used: mpi_utilities

function: simpleconstructorparameters
Description: Constructor for the simple posterior sampling state class which builds the object from a parameter set.
Code lines: 20
Contained by: file posterior_sampling.state.simple.F90
Modules used: input_parameters

function: simpleget
Description: Return the current state.
Code lines: 8
Contained by: file posterior_sampling.state.simple.F90

function: simplemean
Description: Return the mean over state history.
Code lines: 10
Contained by: file posterior_sampling.state.simple.F90
Modules used: galacticus_error

subroutine: simpleparametercountset
Description: Set the number of parameters in this state.
Code lines: 9
Contained by: file posterior_sampling.state.simple.F90

subroutine: simplereset
Description: Reset the state object.
Code lines: 8
Contained by: file posterior_sampling.state.simple.F90

subroutine: simplerestore
Description: Restore the state object.
Code lines: 25
Contained by: file posterior_sampling.state.simple.F90
Modules used: galacticus_error iso_varying_string mpi_utilities

subroutine: `simpleupdate`
Description: Update the current state.
Code lines: 22
Contained by: file posterior_sampling.state.simple.F90

function: `simplevariance`
Description: Return the mean over state history.
Code lines: 10
Contained by: file posterior_sampling.state.simple.F90
Modules used: galacticus_error

file: `posterior_sampling.stopping_criteria.F90`
Description: Contains a module which implements a stopping criteria class for constraint simulations.
Code lines: 41

module: `posterior_sampling_stopping_criteria`
Description: Implements a stopping criteria class for constraint simulations.
Code lines: 19
Contained by: file posterior_sampling.stopping_criteria.F90
Modules used: posterior_sampling_convergence, posterior_sampling_state
Used by: subroutine galacticus_function_classes_destroy, subroutine galacticus_state_retrieve, subroutine galacticus_state_store, file posterior_sampling.simulation.differential_evolution.F90, file posterior_sampling.simulation.particle_swarm.F90

file: `posterior_sampling.stopping_criteria.correlation_length_count.F90`
Description: Implementation of a posterior sampling stopping class which stops after a given number of correlation legnths.
Code lines: 105
Modules used: posterior_sampling_convergence

function: `correlationlengthconstructorinternal`
Description: Internal constructor for the `correlationLength` posterior sampling stopping class.
Code lines: 10
Contained by: file posterior_sampling.stopping_criteria.correlation_length_count.F90
Modules used: input_parameters

function: `correlationlengthconstructorparameters`
Description: Constructor for the `correlationLength` posterior sampling stopping class which builds the object from a parameter set.
Code lines: 21
Contained by: file posterior_sampling.stopping_criteria.correlation_length_count.F90
Modules used: input_parameters

subroutine: `correlationlengthdestructor`
Description: Destructor for the `correlationLength` posterior sampling stopping class.
Code lines: 7
Contained by: file `posterior_sampling.stopping_criteria.correlation_length_count.F90`

function: `correlationlengthstop`
Description: Returns true if the posterior sampling should stop.
Code lines: 16
Contained by: file `posterior_sampling.stopping_criteria.correlation_length_count.F90`
Modules used: `galacticus_error`

interface: `posteriorsamplestoppingcriterioncorrelationlength`
Description: Constructors for the `correlationLength` posterior sampling stopping criterion class.
Code lines: 4
Contained by: file `posterior_sampling.stopping_criteria.correlation_length_count.F90`

file: `posterior_sampling.stopping_criteria.never.F90`
Description: Implementation of a posterior sampling stopping class which never stops.
Code lines: 60

function: `neverconstructorparameters`
Description: Constructor for the `never` posterior sampling stopping class which builds the object from a parameter set.
Code lines: 10
Contained by: file `posterior_sampling.stopping_criteria.never.F90`
Modules used: `input_parameters`

function: `neverstop`
Description: Returns true if the posterior sampling should stop (which it never should).
Code lines: 9
Contained by: file `posterior_sampling.stopping_criteria.never.F90`

interface: `posteriorsamplestoppingcriterionnever`
Description: Constructors for the `never` posterior sampling convergence class.
Code lines: 3
Contained by: file `posterior_sampling.stopping_criteria.never.F90`

file: `posterior_sampling.stopping_criteria.step_count.F90`
Description: Implementation of a posterior sampling stopping class which stepCount stops.
Code lines: 97
Modules used: `posterior_sampling_convergence`

interface: `posteriorsamplestoppingcriterionstepcount`
Description: Constructors for the `stepCount` posterior sampling convergence class.
Code lines: 4
Contained by: file `posterior_sampling.stopping_criteria.step_count.F90`

function: `stepcountconstructorinternal`
Description: Internal constructor for the `stepCount` posterior sampling stopping class.
Code lines: 10
Contained by: file posterior_sampling.stopping_criteria.step_count.F90
Modules used: input_parameters

function: `stepcountconstructorparameters`
Description: Constructor for the `stepCount` posterior sampling stopping class which builds the object from a parameter set.
Code lines: 21
Contained by: file posterior_sampling.stopping_criteria.step_count.F90
Modules used: input_parameters

subroutine: `stepcountdestructor`
Description: Destructor for the `stepCount` posterior sampling stopping class.
Code lines: 7
Contained by: file posterior_sampling.stopping_criteria.step_count.F90

function: `stepcountstop`
Description: Returns true if the posterior sampling should stop.
Code lines: 8
Contained by: file posterior_sampling.stopping_criteria.step_count.F90

file: `radiation.F90`
Description: Contains a module which implements a class to describe radiation fields.
Code lines: 104

module: `radiation_fields`
Description: Implements a class to describe radiation fields.
Code lines: 82
Contained by: file radiation.F90
Modules used: galacticus_nodes
Used by:
file accretion.halo.simple.F90
module chemical_reaction_rates
subroutine hydrogennetworkrateh2_-electron_to_2h_electron
subroutine hydrogennetworkrateh2_-gamma_to_2h
subroutine hydrogennetworkrateh2_-gamma_to_h2plus_electron
subroutine hydrogennetworkrateh2_-gamma_to_h2star_to_2h
subroutine hydrogennetworkrateh2_h_to_-3h
subroutine hydrogennetworkrateh2_-hplus_to_h2plus_h
subroutine hydrogennetworkrateh2plus_-electron_to_2h
subroutine hydrogennetworkrateh2plus_-gamma_to_2hplus_electron
subroutine hydrogennetworkrateh2plus_-gamma_to_h_hplus
subroutine hydrogennetworkrateh2plus_-h_to_h2_hplus
subroutine hydrogennetworkrateh2plus_-hminus_to_h2_h
subroutine hydrogennetworkrateh_-electron_to_hminus_photon
subroutine hydrogennetworkrateh_-electron_to_hplus_2electron
subroutine hydrogennetworkrateh_gamma_-to_hplus_electron

subroutine hydrogennetworkrateh_-hminus_to_h2_electron
subroutine hydrogennetworkrateh_hplus_-to_h2plus_photon
subroutine hydrogennetworkratehminus_-electron_to_h_2electron
subroutine hydrogennetworkratehminus_-gamma_to_h_electron
subroutine hydrogennetworkratehminus_-h_to_2h_electron
subroutine hydrogennetworkratehminus_-hplus_to_2h
subroutine hydrogennetworkratehminus_-hplus_to_h2plus_electron
subroutine hydrogennetworkratehplus_-electron_to_h_photon
subroutine hydrogennetworkrates
subroutine ciefilechemicaldensities
function ciefileelectrondensity
function ciefileelectrondensitydensitylogslope
function ciefileelectrondensitytemperaturelogslope
module chemical_states
function ciefilecoolingfunction
function ciefilecoolingfunctiondensitylogslope
function ciefilecoolingfunctiontemperaturelogslope
function cmbcomptoncoolingfunction
function cmbcomptoncoolingfunctiondensitylogslope
function cmbcomptoncoolingfunctiontemperaturelogslope
function cmbcomptontemperature
module cooling_functions
function molecularhydrogengallipallacoolingfunction
function molecularhydrogengallipallacoolingfunctiondensitylogslope
function molecularhydrogengallipallacoolingfunctiontemperaturelogslope
function summationcoolingfunction
function summationcoolingfunctiondensitylogslope
function summationcoolingfunctiontemperaturelogslope
file cooling.cooling_radius.beta_-profile.F90
file cooling.cooling_radius.simple.F90
module cooling_times
subroutine galacticus_function_-classes_destroy
subroutine galacticus_state_retrieve
subroutine galacticus_state_store
function icmszextract
function icmxrayluminosityextract
module node_component_hot_halo_cold_-mode
module node_component_hot_halo_-standard
program test_cooling_functions
file universe.operators.intergalactic_-medium_state_evolve.F90

function: `crosssectionintegrand`

Description: Integrand function use in integrating a radiation field over a cross section function.
Code lines: 10
Contained by: module radiation_fields

function: `radiationfieldintegrateovercrosssection_`

Description: Integrate the photon number of the radiation field over a given cross-section function (which should return the cross section in units of cm^2), i.e.:

$$\frac{4\pi}{\mathrm{h}} \int_{\lambda_1}^{\lambda_2} \sigma(\lambda) j_\nu(\lambda) \frac{\mathrm{d}\lambda}{\lambda}, \tag{18.33}$$

where j_ν is the flux of energy per unit area per unit solid angle and per unit frequency.

Code lines: 29

Contained by: module radiation_fields

Modules used: fgsl, numerical_constants_physical, numerical_constants_units, numerical_integration

file: `radiation.black_body.F90`
Description: Implements a class for blackbody radiation fields.
Code lines: 103

function: `blackbodyconstructorinternal`
Description: Internal constructor for the `blackBody` radiation field class.
Code lines: 8
Contained by: file `radiation.black_body.F90`

function: `blackbodyconstructorparameters`
Description: Constructor for the `blackBody` radiation field class which takes a parameter list as input.
Code lines: 19
Contained by: file `radiation.black_body.F90`
Modules used: `input_parameters`

function: `blackbodyflux`
Description: Return the flux of a blackBody radiation field.
Code lines: 12
Contained by: file `radiation.black_body.F90`
Modules used: `numerical_constants_units` `thermodynamics_radiation`

function: `blackbodytemperature`
Description: Return the temperature of a black body radiation field.
Code lines: 7
Contained by: file `radiation.black_body.F90`

interface: `radiationfieldblackbody`
Description: Constructors for the `blackBody` radiation field class.
Code lines: 4
Contained by: file `radiation.black_body.F90`

file: `radiation.cosmic_microwave_background.F90`
Description: Implements a class for the cosmic microwave background radiation field.
Code lines: 98
Modules used: `cosmology_functions`

function: `cosmicmicrowavebackgroundconstructorinternal`
Description: Internal constructor for the `cosmicMicrowaveBackground` radiation field class.
Code lines: 9
Contained by: file `radiation.cosmic_microwave_background.F90`

function: `cosmicmicrowavebackgroundconstructorparameters`
Description: Constructor for the `cosmicMicrowaveBackground` radiation field class which takes a parameter list as input.
Code lines: 13
Contained by: file `radiation.cosmic_microwave_background.F90`
Modules used: `input_parameters`

subroutine: `cosmicmicrowavebackgrounddestructor`
Description: Destructor for the `cosmicMicrowaveBackground` radiation field class.
Code lines: 7
Contained by: file radiation.cosmic_microwave_background.F90

subroutine: `cosmicmicrowavebackgroundtimeset`
Description: Set the time (and temperature) of the cosmic microwave background radiation field.
Code lines: 9
Contained by: file radiation.cosmic_microwave_background.F90

interface: `radiationfieldcosmicmicrowavebackground`
Description: Constructors for the `cosmicMicrowaveBackground` radiation field class.
Code lines: 4
Contained by: file radiation.cosmic_microwave_background.F90

file: `radiation.intergalactic_background.F90`
Description: Implements a class for intergalactic background light.
Code lines: 49

type: `radiationfieldintergalacticbackground`
Description: A radiation field class for intergalactic background light.
Code lines: 14
Contained by: file radiation.intergalactic_background.F90

file: `radiation.intergalactic_background.file.F90`
Description: Implements a class for intergalactic background light.
Code lines: 209
Modules used: cosmology_functions, fgsl, iso_c_binding

function: `intergalacticbackgroundfileconstructorinternal`
Description: Internal constructor for the `intergalacticBackgroundFile` radiation field class.
Code lines: 70
Contained by: file radiation.intergalactic_background.file.F90
Modules used: array_utilities, fox_dom, galacticus_error, galacticus_paths, io_xml, memory_management

function: `intergalacticbackgroundfileconstructorparameters`
Description: Constructor for the `intergalacticBackgroundFile` radiation field class which takes a parameter list as input.
Code lines: 21
Contained by: file radiation.intergalactic_background.file.F90
Modules used: input_parameters

subroutine: `intergalacticbackgroundfiledestructor`
Description: Destructor for the `intergalacticBackgroundFile` radiation field class.
Code lines: 7

Contained by: file radiation.intergalactic_background.file.F90

function: intergalacticbackgroundfileflux
Description: Return the flux in the intergalactic background radiation field.
Code lines: 23
Contained by: file radiation.intergalactic_background.file.F90
Modules used: numerical_interpolation

subroutine: intergalacticbackgroundfiletimeset
Description: Set the time of the intergalactic backgroun radiation field.
Code lines: 12
Contained by: file radiation.intergalactic_background.file.F90
Modules used: numerical_interpolation

interface: radiationfieldintergalacticbackgroundfile
Description: Constructors for the intergalacticBackgroundFile radiation field class.
Code lines: 4
Contained by: file radiation.intergalactic_background.file.F90

file: radiation.intergalactic_background.internal.F90
Description: Implements a class for intergalactic background light which computes the background internally.
Code lines: 571
Modules used:

accretion_disk_spectra	atomic_cross_sections_ionization_photo
cosmology_functions	cosmology_parameters
fgsl	intergalactic_medium_state
output_times	stellar_population_selectors

subroutine: intergalacticbackgroundinternalautohook
Code lines: 10
Contained by: file radiation.intergalactic_background.internal.F90
Modules used: events_hooks

function: intergalacticbackgroundinternalconstructorinternal
Description: Internal constructor for the intergalacticBackgroundInternal radiation field class.
Code lines: 52
Contained by: file radiation.intergalactic_background.internal.F90
Modules used: memory_management numerical_ranges

function: intergalacticbackgroundinternalconstructorparameters
Description: Constructor for the intergalacticBackgroundInternal radiation field class which takes a parameter list as input.
Code lines: 81
Contained by: file radiation.intergalactic_background.internal.F90
Modules used: input_parameters

subroutine: intergalacticbackgroundinternaldestructor
Description: Destructor for the intergalacticBackgroundInternal radiation field class.

Code lines: 13
Contained by: file radiation.intergalactic_background.internal.F90

function: intergalacticbackgroundinternalflux
Description: Return the flux in the internally-computed intergalatic background.
Code lines: 45
Contained by: file radiation.intergalactic_background.internal.F90
Modules used: galacticus_error, iso_c_binding, numerical_constants_astronomical, numerical_constants_atomic, numerical_constants_physical, numerical_constants_units, numerical_interpolation

function: intergalacticbackgroundinternalodes
Description: Evaluates the ODEs controlling the evolution of cosmic background radiation.
Code lines: 33
Contained by: file radiation.intergalactic_background.internal.F90
Modules used: fgsl, numerical_constants_astronomical, numerical_constants_atomic, numerical_constants_physical, numerical_constants_units, ode_solver_error_codes

type: intergalacticbackgroundinternalstate
Description: Class used to store the state of the intergalactic background radiation field for the internal solver. This will be stored as an attribute of the universe object.
Code lines: 5
Contained by: file radiation.intergalactic_background.internal.F90

subroutine: intergalacticbackgroundinternaltimeset
Description: Set the epoch.
Code lines: 8
Contained by: file radiation.intergalactic_background.internal.F90

subroutine: intergalacticbackgroundinternaluniversepreevolve
Description: Attach an initial event to the universe to cause the background radiation update function to be called.
Code lines: 32
Contained by: file radiation.intergalactic_background.internal.F90
Modules used: galacticus_error, galacticus_nodes

function: intergalacticbackgroundinternalupdate
Description: Update the radiation background for a given universe.
Code lines: 200
Contained by: file radiation.intergalactic_background.internal.F90
Modules used: abundances_structure, arrays_search, fgsl, fodeiv2, galactic_structure_options, galacticus_display, galacticus_error, galacticus_hdf5, galacticus_nodes, io_hdf5, iso_c_binding, iso_varying_string

kind_numbers
merger_tree_walkers
numerical_constants_math
numerical_constants_physical
numerical_constants_prefixes
numerical_constants_units
numerical_integration
odeiv2_solver
stellar_population_spectra
stellar_populations

function: `stellarspectraconvolution`
Description: Integrand for convolution of stellar spectra.
Code lines: 9
Contained by: function `intergalacticbackgroundinternalupdate`

interface: `radiationfieldintergalacticbackgroundinternal`
Description: Constructors for the `intergalacticBackgroundInternal` radiation field class.
Code lines: 4
Contained by: file `radiation.intergalactic_background.internal.F90`

file: `radiation.null.F90`
Description: Implements a class for null radiation fields.
Code lines: 61

function: `nullconstructorparameters`
Description: Constructor for the `null` radiation field class which takes a parameter list as input.
Code lines: 10
Contained by: file `radiation.null.F90`
Modules used: `input_parameters`

function: `nullflux`
Description: Return the flux of a null radiation field.
Code lines: 10
Contained by: file `radiation.null.F90`

interface: `radiationfieldnull`
Description: Constructors for the `null` radiation field class.
Code lines: 3
Contained by: file `radiation.null.F90`

file: `radiation.summation.F90`
Description: Implements a radiation field class which sums over other radiation fields.
Code lines: 174

type: `radiationfieldlist`
Code lines: 3
Contained by: file `radiation.summation.F90`

interface: `radiationfieldsummation`
Description: Constructors for the "summation" radiation field class.
Code lines: 4
Contained by: file `radiation.summation.F90`

function: `summationconstructorinternal`
Description: Internal constructor for the summation radiation field class.
Code lines: 14
Contained by: file `radiation.summation.F90`

function: `summationconstructorparameters`
Description: Constructor for the "summation" radiation field class which takes a parameter set as input.
Code lines: 22
Contained by: file `radiation.summation.F90`
Modules used: `input_parameters`

subroutine: `summationdeepcopy`
Description: Perform a deep copy for the `summation` radiation field class.
Code lines: 31
Contained by: file `radiation.summation.F90`
Modules used: `galacticus_error`

subroutine: `summationdestructor`
Description: Destructor for the summation radiation field class.
Code lines: 16
Contained by: file `radiation.summation.F90`

function: `summationflux`
Description: Implement a summation radiation field.
Code lines: 15
Contained by: file `radiation.summation.F90`

function: `summationlist`
Description: Return a list of all components for the `summation` radiation field class.
Code lines: 8
Contained by: file `radiation.summation.F90`

file: `ram_pressure_stripping.mass_loss_rate.disks.F90`
Description: Contains a module which provides a class that implements ram pressure stripping in disks.
Code lines: 40

module: `ram_pressure_stripping_mass_loss_rate_disks`
Description: Provides a class that implements calculations of ram pressure stripping in disks.
Code lines: 18
Contained by: file `ram_pressure_stripping.mass_loss_rate.disks.F90`
Modules used: `galacticus_nodes`
Used by:
- subroutine `galacticus_function_classes_destroy`
- subroutine `galacticus_state_retrieve`
- subroutine `galacticus_state_store`
- module `node_component_disk_standard`

file: `ram_pressure_stripping.mass_loss_rate.disks.null.F90`
Description: Implementation of a null ram pressure stripping of disks class.
Code lines: 61

function: `nullconstructorparameters`
Description: Constructor for the `null` timescale for star formation feedback in disks class which takes a parameter set as input.
Code lines: 11
Contained by: file `ram_pressure_stripping.mass_loss_rate.disks.null.F90`
Modules used: `input_parameters`

function: `nullratemassloss`
Description: Returns a zero mass loss rate due to ram pressure stripping.
Code lines: 9
Contained by: file `ram_pressure_stripping.mass_loss_rate.disks.null.F90`

interface: `rampressurestrippingdisksnull`
Description: Constructors for the `null` model of ram pressure stripping of disks class.
Code lines: 3
Contained by: file `ram_pressure_stripping.mass_loss_rate.disks.null.F90`

file: `ram_pressure_stripping.mass_loss_rate.disks.simple.F90`
Description: Implementation of a simple ram pressure stripping of disks class.
Code lines: 144
Modules used: `hot_halo_ram_pressure_forces`

interface: `rampressurestrippingdiskssimple`
Description: Constructors for the `simple` model of ram pressure stripping of disks class.
Code lines: 4
Contained by: file `ram_pressure_stripping.mass_loss_rate.disks.simple.F90`

function: `simpleconstructorinternal`
Description: Internal constructor for the `simple` model of ram pressure stripping of disks class.
Code lines: 9
Contained by: file `ram_pressure_stripping.mass_loss_rate.disks.simple.F90`

function: `simpleconstructorparameters`
Description: Constructor for the `simple` timescale for star formation feedback in disks class which takes a parameter set as input.
Code lines: 23
Contained by: file `ram_pressure_stripping.mass_loss_rate.disks.simple.F90`
Modules used: `input_parameters`

subroutine: `simpledestructor`
Description: Destructor for the `simple` model of ram pressure stripping of disks class.
Code lines: 7
Contained by: file `ram_pressure_stripping.mass_loss_rate.disks.simple.F90`

function: `simpleratemassloss`

Description: Computes the mass loss rate from disks due to ram pressure stripping assuming a simple model. Specifically, the mass loss rate is

$$\dot{M}_{\rm gas} = -\alpha M_{\rm gas}/\tau_{\rm disk}, \tag{18.34}$$

where

$$\alpha = F_{\rm ram}/F_{\rm gravity}, \tag{18.35}$$

$F_{\rm ram}$ is the ram pressure force from the hot halo (see §12.23), and

$$F_{\rm gravity} = 2\pi {\rm G}\Sigma_{\rm gas}(r_{1/2})\Sigma_{\rm total}(r_{1/2}) \tag{18.36}$$

is the gravitational restoring force in the disk at the half-mass radius, $r_{1/2}$.

Code lines: 54

Contained by: file `ram_pressure_stripping.mass_loss_rate.disks.simple.F90`

Modules used: galactic_structure_options, galactic_structure_surface_densities, galacticus_nodes, numerical_constants_astronomical, numerical_constants_physical

file: ram_pressure_stripping.mass_loss_rate.spheroids.F90
Description: Contains a module which provides a class that implements ram pressure stripping in spheroids.
Code lines: 40

module: ram_pressure_stripping_mass_loss_rate_spheroids
Description: Provides a class that implements calculations of ram pressure stripping in spheroids.
Code lines: 18
Contained by: file ram_pressure_stripping.mass_loss_rate.spheroids.F90
Modules used: galacticus_nodes
Used by: subroutine galacticus_function_-classes_destroy, subroutine galacticus_state_retrieve, subroutine galacticus_state_store, module node_component_spheroid_-standard

file: ram_pressure_stripping.mass_loss_rate.spheroids.null.F90
Description: Implementation of a null ram pressure stripping of spheroids class.
Code lines: 61

function: nullconstructorparameters
Description: Constructor for the null timescale for star formation feedback in spheroids class which takes a parameter set as input.
Code lines: 11
Contained by: file ram_pressure_stripping.mass_loss_rate.spheroids.null.F90
Modules used: input_parameters

function: nullratemassloss
Description: Returns a zero mass loss rate due to ram pressure stripping.
Code lines: 9
Contained by: file ram_pressure_stripping.mass_loss_rate.spheroids.null.F90

interface: rampressurestrippingspheroidsnull
Description: Constructors for the null model of ram pressure stripping of spheroids class.
Code lines: 3
Contained by: file ram_pressure_stripping.mass_loss_rate.spheroids.null.F90

file: ram_pressure_stripping.mass_loss_rate.spheroids.simple.F90
Description: Implementation of a simple ram pressure stripping of spheroids class.
Code lines: 145
Modules used: hot_halo_ram_pressure_forces

interface: rampressurestrippingspheroidssimple
Description: Constructors for the simple model of ram pressure stripping of spheroids class.
Code lines: 4
Contained by: file ram_pressure_stripping.mass_loss_rate.spheroids.simple.F90

function: `simpleconstructorinternal`
Description: Internal constructor for the `simple` model of ram pressure stripping of spheroids class.
Code lines: 9
Contained by: file `ram_pressure_stripping.mass_loss_rate.spheroids.simple.F90`

function: `simpleconstructorparameters`
Description: Constructor for the `simple` timescale for star formation feedback in spheroids class which takes a parameter set as input.
Code lines: 23
Contained by: file `ram_pressure_stripping.mass_loss_rate.spheroids.simple.F90`
Modules used: `input_parameters`

subroutine: `simpledestructor`
Description: Destructor for the `simple` model of ram pressure stripping of spheroids class.
Code lines: 7
Contained by: file `ram_pressure_stripping.mass_loss_rate.spheroids.simple.F90`

function: `simpleratemassloss`
Description: Computes the mass loss rate from spheroids due to ram pressure stripping assuming a simple model. Specifically, the mass loss rate is

$$\dot{M}_{\rm gas} = -\alpha M_{\rm gas}/\tau_{\rm spheroid}, \tag{18.37}$$

where

$$\alpha = F_{\rm ram}/F_{\rm gravity}, \tag{18.38}$$

$F_{\rm ram}$ is the ram pressure force from the hot halo (see §12.23), and

$$F_{\rm gravity} = \frac{4}{3}\rho_{\rm gas}(r_{1/2})\frac{{\rm G}M_{\rm total}(r_{1/2})}{r_{1/2}} \tag{18.39}$$

is the gravitational restoring force in the spheroid at the half-mass radius, $r_{1/2}$ [Takeda et al., 1984].
Code lines: 55
Contained by: file `ram_pressure_stripping.mass_loss_rate.spheroids.simple.F90`
Modules used: `galactic_structure_densities` `galactic_structure_enclosed_masses` `galactic_structure_options` `galacticus_nodes` `numerical_constants_astronomical` `numerical_constants_physical`

file: `satellites.dynamical_friction.acceleration.Chandrasekhar1943.F90`
Description: Implementation of a satellite dynamical friction class which uses the model of Chandrasekhar [1943].
Code lines: 136
Modules used: `dark_matter_halo_scales` `dark_matter_profiles_dmo`

function: `chandrasekhar1943acceleration`
Description: Return an acceleration for satellites due to dynamical friction using the formulation of Chandrasekhar [1943].
Code lines: 37
Contained by: file `satellites.dynamical_friction.acceleration.Chandrasekhar1943.F90`

Modules used: error_functions, galactic_structure_densities, galactic_structure_options, galacticus_nodes, numerical_constants_astronomical, numerical_constants_math, numerical_constants_physical, numerical_constants_prefixes, vectors

function: chandrasekhar1943constructorinternal
Description: Internal constructor for the chandrasekhar1943 satellite dynamical friction class.
Code lines: 10
Contained by: file satellites.dynamical_friction.acceleration.Chandrasekhar1943.F90

function: chandrasekhar1943constructorparameters
Description: Constructor for the chandrasekhar1943 satellite dynamical friction class which builds the object from a parameter set.
Code lines: 25
Contained by: file satellites.dynamical_friction.acceleration.Chandrasekhar1943.F90
Modules used: input_parameters

subroutine: chandrasekhar1943destructor
Description: Destructor for the simple cooling radius class.
Code lines: 8
Contained by: file satellites.dynamical_friction.acceleration.Chandrasekhar1943.F90

interface: satellitedynamicalfrictionchandrasekhar1943
Description: Constructors for the chandrasekhar1943 satellite dynamical friction class.
Code lines: 4
Contained by: file satellites.dynamical_friction.acceleration.Chandrasekhar1943.F90

file: satellites.dynamical_friction.acceleration.F90
Description: Contains a module that implements calculations of the acceleration due to dynamical friction for satellites.
Code lines: 42

module: satellite_dynamical_friction
Description: Implements calculations of dynamical friction for satellites.
Code lines: 18
Contained by: file satellites.dynamical_friction.acceleration.F90
Modules used: galacticus_nodes
Used by: subroutine galacticus_function_classes_destroy, subroutine galacticus_state_retrieve, subroutine galacticus_state_store, module node_component_satellite_orbiting

file: satellites.dynamical_friction.acceleration.zero.F90
Description: Implementation of a zero acceleration satellite dynamical friction class.
Code lines: 61

interface: satellitedynamicalfrictionzero
Description: Constructors for the zero satellite dynamical friction class.

Code lines: 3
Contained by: file satellites.dynamical_friction.acceleration.zero.F90

function: zeroacceleration
Description: Return a zero acceleration for satellites due to dynamical friction.
Code lines: 10
Contained by: file satellites.dynamical_friction.acceleration.zero.F90

function: zeroconstructorparameters
Description: Constructor for the zero satellite dynamical friction class which builds the object from a parameter set.
Code lines: 10
Contained by: file satellites.dynamical_friction.acceleration.zero.F90
Modules used: input_parameters

file: satellites.merging.mass_movements.Baugh2005.F90
Description: Implements a merger mass movements class using the Baugh et al. [2005] model.
Code lines: 140

function: baugh2005constructorinternal
Description: Internal constructor for the baugh2005 merger mass movements.
Code lines: 9
Contained by: file satellites.merging.mass_movements.Baugh2005.F90

function: baugh2005constructorparameters
Description: Constructor for the baugh2005 merger mass movements class which takes a parameter list as input.
Code lines: 44
Contained by: file satellites.merging.mass_movements.Baugh2005.F90
Modules used: input_parameters

subroutine: baugh2005get
Description: Determine how different mass components should be redistributed as the result of a merger according to the model of Baugh et al. [2005].
Code lines: 41
Contained by: file satellites.merging.mass_movements.Baugh2005.F90
Modules used: galactic_structure_enclosed_masses galactic_structure_options

interface: mergermassmovementsbaugh2005
Description: Constructors for the baugh2005 merger mass movements class.
Code lines: 4
Contained by: file satellites.merging.mass_movements.Baugh2005.F90

file: satellites.merging.mass_movements.F90
Description: Contains a module which implements a class for determining how mass is moved around as a consequence of a satellite merging event.
Code lines: 53

module: satellite_merging_mass_movements

Description: Implements a class for determining how mass is moved around as a consequence of a satellite merging event.
Code lines: 30
Contained by: file satellites.merging.mass_movements.F90
Modules used: galacticus_nodes
Used by: subroutine satellite_merging_remnant_compute; subroutine galacticus_function_classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; subroutine node_component_age_statistics_standard_satellite_merging; subroutine node_component_disk_standard_satellite_merging; subroutine node_component_disk_very_simple_satellite_merging; subroutine node_component_inter_output_standard_satellite_merging; subroutine node_component_merging_statistics_major_satellite_merging; subroutine node_component_spheroid_standard_satellite_merging; subroutine node_component_spheroid_very_simple_satellite_merging; file satellites.merging.progenitor_properties.Cole2000.F90; function cole2000halfmassradiusroot; file satellites.merging.progenitor_properties.simple.F90; file satellites.merging.progenitor_properties.standard.F90

file: satellites.merging.mass_movements.simple.F90
Description: Implements a merger mass movements class which uses a simple calculation.
Code lines: 111

interface: mergermassmovementssimple
Description: Constructors for the simple merger mass movements class.
Code lines: 4
Contained by: file satellites.merging.mass_movements.simple.F90

function: simpleconstructorinternal
Description: Internal constructor for the simple merger mass movements class.
Code lines: 9
Contained by: file satellites.merging.mass_movements.simple.F90

function: simpleconstructorparameters
Description: Constructor for the simple merger mass movements class which takes a parameter list as input.
Code lines: 28
Contained by: file satellites.merging.mass_movements.simple.F90
Modules used: input_parameters

subroutine: simpleget
Description: Determine where stars and gas move as the result of a merger event using a simple algorithm.
Code lines: 28
Contained by: file satellites.merging.mass_movements.simple.F90
Modules used: galactic_structure_enclosed_masses; galactic_structure_options

file: satellites.merging.mass_movements.very_simple.F90
Description: Implements a merger mass movements class which uses a simple calculation.
Code lines: 99

interface: mergermassmovementsverysimple
Description: Constructors for the verySimple merger mass movements class.
Code lines: 4
Contained by: file satellites.merging.mass_movements.very_simple.F90

function: verysimpleconstructorinternal
Description: Internal constructor for the verySimple merger mass movements.
Code lines: 8
Contained by: file satellites.merging.mass_movements.very_simple.F90

function: verysimpleconstructorparameters
Description: Constructor for the verySimple merger mass movements class which takes a parameter list as input.
Code lines: 19
Contained by: file satellites.merging.mass_movements.very_simple.F90
Modules used: input_parameters

subroutine: verysimpleget
Description: Determine where stars and gas move as the result of a merger event using a very simple algorithm.
Code lines: 27
Contained by: file satellites.merging.mass_movements.very_simple.F90
Modules used: galactic_structure_enclosed_masses galactic_structure_options

file: satellites.merging.output.F90
Description: Contains a module which outputs data on all merging events which occur.
Code lines: 103

module: satellites_merging_output
Description: Implement outputting of all merging events.
Code lines: 81
Contained by: file satellites.merging.output.F90
Used by: subroutine satellitemergerprocess

subroutine: satellite_merging_output
Description: Outputs properties of merging nodes.
Code lines: 62
Contained by: module satellites_merging_output
Modules used: galacticus_hdf5 galacticus_nodes
input_parameters io_hdf5

file: satellites.merging.progenitor_properties.Cole2000.F90
Description: Implements a merger progenitor properties class which uses the algorithm of Cole et al. [2000].
Code lines: 292

Modules used: satellite_merging_mass_movements

function: cole2000constructorinternal
Description: Internal constructor for the cole2000 merger progenitor properties class.
Code lines: 8
Contained by: file satellites.merging.progenitor_properties.Cole2000.F90

function: cole2000constructorparameters
Description: Constructor for the cole2000 merger progenitor properties class which takes a parameter list as input.
Code lines: 19
Contained by: file satellites.merging.progenitor_properties.Cole2000.F90
Modules used: array_utilities, galacticus_error, galacticus_nodes, input_parameters

subroutine: cole2000destructor
Description: Destructor for the cole2000 merger progenitor properties class.
Code lines: 7
Contained by: file satellites.merging.progenitor_properties.Cole2000.F90

subroutine: cole2000get
Description: Computes various properties of the progenitor galaxies useful for calculations of merger remnant sizes.
Code lines: 175
Contained by: file satellites.merging.progenitor_properties.Cole2000.F90
Modules used: galactic_structure_enclosed_masses, galactic_structure_options, galacticus_error, galacticus_nodes, numerical_constants_physical, root_finder

function: cole2000halfmassradiusroot
Description: Function used in root finding for progenitor galaxy half-mass radii.
Code lines: 25
Contained by: file satellites.merging.progenitor_properties.Cole2000.F90
Modules used: galactic_structure_enclosed_masses, galactic_structure_options, satellite_merging_mass_movements

interface: mergerprogenitorpropertiescole2000
Description: Constructors for the cole2000 merger progenitor properties class.
Code lines: 4
Contained by: file satellites.merging.progenitor_properties.Cole2000.F90

file: satellites.merging.progenitor_properties.F90
Description: Contains a module which implements a class for calculations for progenitor properties for mergers.
Code lines: 46

module: satellite_merging_progenitor_properties
Description: Implements a class for calculations for progenitor properties for mergers.
Code lines: 24
Contained by: file satellites.merging.progenitor_properties.F90

Modules used: galacticus_nodes
Used by: subroutine galacticus_function_classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; file satellites.merging.remnant_sizes.Cole2000.F90; file satellites.merging.remnant_sizes.Covington2008.F90

file: satellites.merging.progenitor_properties.simple.F90
Description: Implements a merger progenitor properties class which uses a simple calculation.
Code lines: 181
Modules used: satellite_merging_mass_movements

interface: mergerprogenitorpropertiessimple
Description: Constructors for the simple merger progenitor properties class.
Code lines: 4
Contained by: file satellites.merging.progenitor_properties.simple.F90

function: simpleconstructorinternal
Description: Internal constructor for the simple merger progenitor properties class.
Code lines: 8
Contained by: file satellites.merging.progenitor_properties.simple.F90

function: simpleconstructorparameters
Description: Constructor for the simple merger progenitor properties class which takes a parameter list as input.
Code lines: 19
Contained by: file satellites.merging.progenitor_properties.simple.F90
Modules used: array_utilities; galacticus_error; galacticus_nodes; input_parameters

subroutine: simpledestructor
Description: Destructor for the simple merger progenitor properties class.
Code lines: 7
Contained by: file satellites.merging.progenitor_properties.simple.F90

subroutine: simpleget
Description: Computes various properties of the progenitor galaxies useful for calculations of merger remnant sizes.
Code lines: 97
Contained by: file satellites.merging.progenitor_properties.simple.F90
Modules used: galactic_structure_enclosed_masses; galactic_structure_options; galacticus_error; galacticus_nodes; numerical_constants_physical

file: satellites.merging.progenitor_properties.standard.F90
Description: Implements a merger progenitor properties class which uses a standard calculation.
Code lines: 236
Modules used: satellite_merging_mass_movements

interface: `mergerprogenitorpropertiesstandard`
Description: Constructors for the `standard` merger progenitor properties class.
Code lines: 4
Contained by: file `satellites.merging.progenitor_properties.standard.F90`

function: `standardconstructorinternal`
Description: Internal constructor for the `standard` merger progenitor properties class.
Code lines: 8
Contained by: file `satellites.merging.progenitor_properties.standard.F90`

function: `standardconstructorparameters`
Description: Constructor for the `standard` merger progenitor properties class which takes a parameter list as input.
Code lines: 18
Contained by: file `satellites.merging.progenitor_properties.standard.F90`
Modules used: `array_utilities` `galacticus_error` `galacticus_nodes` `input_parameters`

subroutine: `standarddestructor`
Description: Destructor for the `standard` merger progenitor properties class.
Code lines: 7
Contained by: file `satellites.merging.progenitor_properties.standard.F90`

subroutine: `standardget`
Description: Computes various properties of the progenitor galaxies useful for calculations of merger remnant sizes.
Code lines: 153
Contained by: file `satellites.merging.progenitor_properties.standard.F90`
Modules used: `galactic_structure_enclosed_masses` `galactic_structure_options` `galacticus_error` `galacticus_nodes` `numerical_constants_physical`

file: `satellites.merging.remnant_sizes.Cole2000.F90`
Description: Implements a merger remnant size class which uses the Cole et al. [2000] algorithm.
Code lines: 191
Modules used: `satellite_merging_progenitor_-properties`

function: `cole2000constructorinternal`
Description: Internal constructor for the `cole2000` merger remnant size class.
Code lines: 9
Contained by: file `satellites.merging.remnant_sizes.Cole2000.F90`

function: `cole2000constructorparameters`
Description: Constructor for the `cole2000` merger remnant size class which takes a parameter list as input.
Code lines: 22
Contained by: file `satellites.merging.remnant_sizes.Cole2000.F90`
Modules used: `input_parameters`

subroutine: `cole2000destructor`
Description: Destructor for the `cole2000` merger remnant size class.
Code lines: 7
Contained by: file `satellites.merging.remnant_sizes.Cole2000.F90`

subroutine: `cole2000get`
Description: Compute the size of the merger remnant for `node` using the Cole et al. [2000] algorithm.
Code lines: 102
Contained by: file `satellites.merging.remnant_sizes.Cole2000.F90`
Modules used: `galactic_structure_enclosed_masses` `galactic_structure_options` `galacticus_display` `galacticus_error` `numerical_comparison` `numerical_constants_physical` `string_handling`

interface: `mergerremnantsizecole2000`
Description: Constructors for the `cole2000` merger remnant size class.
Code lines: 4
Contained by: file `satellites.merging.remnant_sizes.Cole2000.F90`

file: `satellites.merging.remnant_sizes.Covington2008.F90`
Description: Implements a merger remnant size class which uses the Cole et al. [2000] algorithm.
Code lines: 229
Modules used: `dark_matter_halo_scales` `satellite_merging_progenitor_properties`

function: `covington2008constructorinternal`
Description: Internal constructor for the `covington2008` merger remnant size class.
Code lines: 11
Contained by: file `satellites.merging.remnant_sizes.Covington2008.F90`

function: `covington2008constructorparameters`
Description: Constructor for the `covington2008` merger remnant size class which takes a parameter list as input.
Code lines: 34
Contained by: file `satellites.merging.remnant_sizes.Covington2008.F90`
Modules used: `input_parameters`

subroutine: `covington2008destructor`
Description: Destructor for the `covington2008` merger remnant size class.
Code lines: 8
Contained by: file `satellites.merging.remnant_sizes.Covington2008.F90`

subroutine: `covington2008get`
Description: Compute the size of the merger remnant for `node` using the Covington et al. [2008] algorithm.
Code lines: 122
Contained by: file `satellites.merging.remnant_sizes.Covington2008.F90`
Modules used: `galacticus_display` `galacticus_error` `numerical_comparison` `numerical_constants_physical`

string_handling

interface: `mergerremnantsizecovington2008`
Description: Constructors for the `covington2008` merger remnant size class.
Code lines: 4
Contained by: file satellites.merging.remnant_sizes.Covington2008.F90

file: `satellites.merging.remnant_sizes.F90`
Description: Contains a module which implements a class for calculations of merger remnant sizes.
Code lines: 45

module: `satellite_merging_remnant_sizes`
Description: Implements a class for calculations of merger remnant sizes.
Code lines: 23
Contained by: file satellites.merging.remnant_sizes.F90
Modules used: galacticus_nodes
Used by: subroutine satellite_merging_remnant_compute; subroutine galacticus_function_classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; subroutine node_component_spheroid_standard_satellite_merging

file: `satellites.merging.remnant_sizes.null.F90`
Description: Implements a merger remnant size class which takes no action.
Code lines: 60

interface: `mergerremnantsizenull`
Description: Constructors for the `null` merger remnant size class.
Code lines: 3
Contained by: file satellites.merging.remnant_sizes.null.F90

function: `nullconstructorparameters`
Description: Constructor for the `null` merger remnant size class which takes a parameter list as input.
Code lines: 10
Contained by: file satellites.merging.remnant_sizes.null.F90
Modules used: input_parameters

subroutine: `nullget`
Description: Do not compute the size of the merger remnant for `node`.
Code lines: 9
Contained by: file satellites.merging.remnant_sizes.null.F90

file: `satellites.merging.timescale.Boylan-Kolchin2008.F90`
Description: Implements calculations of satellite merging times using the Boylan-Kolchin et al. [2008] method.
Code lines: 154
Modules used: dark_matter_halo_scales; dark_matter_profiles_dmo

function: `boylankolchin2008constructorinternal`

Description: Default constructor for the boylanKolchin2008 satellite merging timescale class.
Code lines: 10
Contained by: file satellites.merging.timescale.Boylan-Kolchin2008.F90

function: boylankolchin2008constructorparameters
Description: A constructor for the boylanKolchin2008 satellite merging timescale class which builds the object from a parameter set.
Code lines: 26
Contained by: file satellites.merging.timescale.Boylan-Kolchin2008.F90
Modules used: input_parameters

subroutine: boylankolchin2008destructor
Description: Destructor for the boylanKolchin2008 satellite merging timescale class.
Code lines: 8
Contained by: file satellites.merging.timescale.Boylan-Kolchin2008.F90

function: boylankolchin2008timeuntilmerging
Description: Return the timescale for merging satellites using the Boylan-Kolchin et al. [2008] method.
Code lines: 57
Contained by: file satellites.merging.timescale.Boylan-Kolchin2008.F90
Modules used: galacticus_error, galacticus_nodes, kepler_orbits, satellite_orbits

interface: satellitemergingtimescalesboylankolchin2008
Description: Constructors for the boylanKolchin2008 satellite merging timescale class.
Code lines: 4
Contained by: file satellites.merging.timescale.Boylan-Kolchin2008.F90

file: satellites.merging.timescale.F90
Description: Contains a module that provides and object that implements satellite merging timescales.
Code lines: 46

module: satellite_merging_timescales
Description: Provides and object that implements satellite merging timescales.
Code lines: 24
Contained by: file satellites.merging.timescale.F90
Modules used: galacticus_nodes, kepler_orbits
Used by: subroutine galacticus_function_classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; file merger_trees.construct.read.F90; file merger_trees.operators.assign_orbits.F90; subroutine node_component_satellite_orbiting_create; module node_component_satellite_standard; module node_component_satellite_very_simple

file: satellites.merging.timescale.Jiang2008.F90
Description: Implements calculations of satellite merging times using the Jiang et al. [2008] method.
Code lines: 165
Modules used: dark_matter_halo_scales, dark_matter_profiles_dmo

function: `jiang2008constructorinternal`
Description: Constructor for the Jiang et al. [2008] merging timescale class.
Code lines: 10
Contained by: file `satellites.merging.timescale.Jiang2008.F90`

function: `jiang2008constructorparameters`
Description: Constructor for the Jiang et al. [2008] merging timescale class which builds the object from a parameter set.
Code lines: 38
Contained by: file `satellites.merging.timescale.Jiang2008.F90`
Modules used: `galacticus_display` `galacticus_error` `galacticus_nodes` `input_parameters`

subroutine: `jiang2008destructor`
Description: Destructor for the `jiang2008` satellite merging timescale class.
Code lines: 8
Contained by: file `satellites.merging.timescale.Jiang2008.F90`

function: `jiang2008timeuntilmerging`
Description: Return the timescale for merging satellites using the Jiang et al. [2008] method.
Code lines: 54
Contained by: file `satellites.merging.timescale.Jiang2008.F90`
Modules used: `galacticus_error` `galacticus_nodes` `satellite_orbits`

interface: `satellitemergingtimescalesjiang2008`
Description: Constructors for the Jiang et al. [2008] merging timescale class.
Code lines: 4
Contained by: file `satellites.merging.timescale.Jiang2008.F90`

file: `satellites.merging.timescale.Lacey-Cole.F90`
Description: Implements a satellite merging timescale class which uses the Lacey and Cole [1993] method.
Code lines: 150
Modules used: `dark_matter_halo_scales`

function: `laceycole1993constructorinternal`
Description: Constructor for the Lacey and Cole [1993] merging timescale class.
Code lines: 9
Contained by: file `satellites.merging.timescale.Lacey-Cole.F90`

function: `laceycole1993constructorparameters`
Description: Constructor for the Lacey and Cole [1993] merging timescale class which builds the object from a parameter set.
Code lines: 22
Contained by: file `satellites.merging.timescale.Lacey-Cole.F90`
Modules used: `input_parameters`

subroutine: `laceycole1993destructor`

Description: Destructor for the `laceyCole1993` satellite merging timescale class.
Code lines: 7
Contained by: file `satellites.merging.timescale.Lacey-Cole.F90`

function: `laceycole1993timeuntilmerging`
Description: Return the timescale for merging satellites using the Lacey and Cole [1993] method.
Code lines: 22
Contained by: file `satellites.merging.timescale.Lacey-Cole.F90`
Modules used: `kepler_orbits`

function: `laceycole1993timeuntilmergingmassdependence`
Description: Return the mass-dependent part of the timescale for merging satellites using the Lacey and Cole [1993] method.
Code lines: 27
Contained by: file `satellites.merging.timescale.Lacey-Cole.F90`
Modules used: `galacticus_nodes`

interface: `satellitemergingtimescaleslaceycole1993`
Description: Constructors for the Lacey and Cole [1993] merging timescale class.
Code lines: 4
Contained by: file `satellites.merging.timescale.Lacey-Cole.F90`

file: `satellites.merging.timescale.Lacey-Cole_Tormen.F90`
Description: Implements calculations of satellite merging times using the Lacey and Cole [1993] method with a parameterization of orbital parameters designed to fit the results of Tormen [1997] as described by Cole et al. [2000].
Code lines: 85

function: `laceycole1993tormenconstructorinternal`
Description: Constructor for the Cole et al. [2000] merging timescale class.
Code lines: 9
Contained by: file `satellites.merging.timescale.Lacey-Cole_Tormen.F90`

function: `laceycole1993tormenconstructorparameters`
Description: Constructor for the Cole et al. [2000] merging timescale class which builds the object from a parameter set.
Code lines: 9
Contained by: file `satellites.merging.timescale.Lacey-Cole_Tormen.F90`
Modules used: `input_parameters`

function: `laceycole1993tormentimeuntilmerging`
Description: Return the timescale for merging satellites using the Lacey and Cole [1993] method with a parameterization of orbital parameters designed to fit the results of Tormen [1997] as described by Cole et al. [2000].
Code lines: 22
Contained by: file `satellites.merging.timescale.Lacey-Cole_Tormen.F90`

interface: `satellitemergingtimescaleslaceycole1993tormen`
Description: Constructors for the Cole et al. [2000] merging timescale class.

Code lines: 4
Contained by: file satellites.merging.timescale.Lacey-Cole_Tormen.F90

file: satellites.merging.timescale.Villalobos_2013.F90
Description: Implements a satellite merging timescale class which applies the Villalobos et al. [2013] modifier to another selected satellite merging time class.
Code lines: 117
Modules used: cosmology_functions

interface: satellitemergingtimescalesvillalobos2013
Description: Constructors for the Lacey and Cole [1993] merging timescale class.
Code lines: 4
Contained by: file satellites.merging.timescale.Villalobos_2013.F90

function: villalobos2013constructorinternal
Description: Constructor for the Lacey and Cole [1993] merging timescale class.
Code lines: 10
Contained by: file satellites.merging.timescale.Villalobos_2013.F90

function: villalobos2013constructorparameters
Description: Constructor for the Lacey and Cole [1993] merging timescale class which builds the object from a parameter set.
Code lines: 27
Contained by: file satellites.merging.timescale.Villalobos_2013.F90
Modules used: input_parameters

subroutine: villalobos2013destructor
Description: Destructor for the villalobos2013 satellite merging timescale class.
Code lines: 8
Contained by: file satellites.merging.timescale.Villalobos_2013.F90

function: villalobos2013timeuntilmerging
Description: Return the timescale for merging satellites using the Villalobos et al. [2013] method.
Code lines: 18
Contained by: file satellites.merging.timescale.Villalobos_2013.F90
Modules used: cosmology_functions galacticus_nodes kepler_orbits

file: satellites.merging.timescale.Wetzel-White.F90
Description: Implements a satellite merging timescale class which uses the Wetzel and White [2010] method.
Code lines: 114
Modules used: cosmology_functions

interface: satellitemergingtimescaleswetzelwhite2010
Description: Constructors for the Lacey and Cole [1993] merging timescale class.
Code lines: 4
Contained by: file satellites.merging.timescale.Wetzel-White.F90

function: `wetzelwhite2010constructorinternal`
Description: Constructor for the Wetzel and White [2010] merging timescale class.
Code lines: 9
Contained by: file `satellites.merging.timescale.Wetzel-White.F90`

function: `wetzelwhite2010constructorparameters`
Description: Constructor for the Wetzel and White [2010] merging timescale class which builds the object from a parameter set.
Code lines: 22
Contained by: file `satellites.merging.timescale.Wetzel-White.F90`
Modules used: `input_parameters`

subroutine: `wetzelwhite2010destructor`
Description: Destructor for the `wetzelWhite2010` satellite merging timescale class.
Code lines: 7
Contained by: file `satellites.merging.timescale.Wetzel-White.F90`

function: `wetzelwhite2010timeuntilmerging`
Description: Return the timescale for merging satellites using the Wetzel and White [2010] method.
Code lines: 23
Contained by: file `satellites.merging.timescale.Wetzel-White.F90`
Modules used: `galacticus_nodes` `kepler_orbits`

file: `satellites.merging.timescale.infinite.F90`
Description: Implements a satellite merging timescale class in which merging timescales are always infinite.
Code lines: 65

function: `infiniteconstructorparameters`
Description: A constructor for the `infinite` satellite merging timescale class which builds the object from a parameter set.
Code lines: 11
Contained by: file `satellites.merging.timescale.infinite.F90`
Modules used: `input_parameters`

function: `infinitetimeuntilmerging`
Description: Return a infinite timescale for satellite merging.
Code lines: 10
Contained by: file `satellites.merging.timescale.infinite.F90`

interface: `satellitemergingtimescalesinfinite`
Description: Constructors for the `infinite` satellite merging timescale class.
Code lines: 3
Contained by: file `satellites.merging.timescale.infinite.F90`

file: `satellites.merging.timescale.preset.F90`
Description: Implements a satellite merging timescale class which uses preset values for the timescale.
Code lines: 67

function: `presetconstructorparameters`
Description: A constructor for the `preset` satellite merging timescale class which builds the object from a parameter set.
Code lines: 11
Contained by: file `satellites.merging.timescale.preset.F90`
Modules used: `input_parameters`

function: `presettimeuntilmerging`
Description: Return the timescale for merging satellites using the preset value.
Code lines: 15
Contained by: file `satellites.merging.timescale.preset.F90`
Modules used: `galacticus_nodes` `kepler_orbits`

interface: `satellitemergingtimescalespreset`
Description: Constructors for the `preset` satellite merging timescale class.
Code lines: 3
Contained by: file `satellites.merging.timescale.preset.F90`

file: `satellites.merging.timescale.random.F90`
Description: Implements calculations of satellite merging times that are chosen to occur randomly between snapshots.
Code lines: 81

function: `randomconstructorparameters`
Description: A constructor for the `random` satellite merging timescale class which builds the object from a parameter set.
Code lines: 11
Contained by: file `satellites.merging.timescale.random.F90`
Modules used: `input_parameters`

function: `randomtimeuntilmerging`
Description: Return a randomly chosen timescale for merging satellites.
Code lines: 26
Contained by: file `satellites.merging.timescale.random.F90`
Modules used: `galacticus_nodes` `kepler_orbits` `pseudo_random` `satellite_orbits`

interface: `satellitemergingtimescalesrandom`
Description: Constructors for the `random` satellite merging timescale class.
Code lines: 3
Contained by: file `satellites.merging.timescale.random.F90`

file: `satellites.merging.timescale.zero.F90`
Description: Implements a satellite merging timescale class in which merging timescales are always zero.
Code lines: 62

interface: `satellitemergingtimescaleszero`
Description: Constructors for the `zero` satellite merging timescale class.

Code lines: 3
Contained by: file satellites.merging.timescale.zero.F90

function: zeroconstructorparameters
Description: A constructor for the zero satellite merging timescale class which builds the object from a parameter set.
Code lines: 11
Contained by: file satellites.merging.timescale.zero.F90
Modules used: input_parameters

function: zerotimeuntilmerging
Description: Return a zero timescale for satellite merging.
Code lines: 10
Contained by: file satellites.merging.timescale.zero.F90

file: satellites.merging.virial_orbits.Benson2005.F90
Description: An implementation of virial orbits using the Benson [2005] orbital parameter distribution.
Code lines: 222
Modules used: cosmology_functions dark_matter_halo_scales

function: benson2005constructorinternal
Description: Internal constructor for the benson2005 virial orbits class.
Code lines: 11
Contained by: file satellites.merging.virial_orbits.Benson2005.F90

function: benson2005constructorparameters
Description: Constructor for the benson2005 virial orbits class which takes a parameter set as input.
Code lines: 16
Contained by: file satellites.merging.virial_orbits.Benson2005.F90
Modules used: input_parameters

function: benson2005densitycontrastdefinition
Description: Return a virial density contrast object defining that used in the definition of Benson [2005] virial orbits.
Code lines: 8
Contained by: file satellites.merging.virial_orbits.Benson2005.F90

subroutine: benson2005destructor
Description: Destructor for the benson2005 virial orbits class.
Code lines: 9
Contained by: file satellites.merging.virial_orbits.Benson2005.F90

function: benson2005orbit
Description: Return benson2005 orbital parameters for a satellite.
Code lines: 61
Contained by: file satellites.merging.virial_orbits.Benson2005.F90
Modules used: dark_matter_profile_mass_definitions galacticus_error galacticus_nodes

function: `benson2005velocitytangentialmagnitudemean`
Description: Return the mean magnitude of the tangential velocity.
Code lines: 17
Contained by: file satellites.merging.virial_orbits.Benson2005.F90
Modules used: dark_matter_profile_mass_definitions galacticus_nodes

function: `benson2005velocitytangentialvectormean`
Description: Return the mean of the vector tangential velocity.
Code lines: 12
Contained by: file satellites.merging.virial_orbits.Benson2005.F90
Modules used: galacticus_error

function: `benson2005velocitytotalrootmeansquared`
Description: Return the mean magnitude of the tangential velocity.
Code lines: 17
Contained by: file satellites.merging.virial_orbits.Benson2005.F90
Modules used: dark_matter_profile_mass_definitions galacticus_nodes

interface: `virialorbitbenson2005`
Description: Constructors for the `benson2005` virial orbit class.
Code lines: 4
Contained by: file satellites.merging.virial_orbits.Benson2005.F90

file: `satellites.merging.virial_orbits.F90`
Description: Contains a module which provides a class implementing satellite orbital parameters at virial radius crossing.
Code lines: 67

module: `virial_orbits`
Description: Provides a class implementing satellite orbital parameters at virial radius crossing.
Code lines: 45
Contained by: file satellites.merging.virial_orbits.F90
Modules used: galacticus_nodes kepler_orbits virial_density_contrast
Used by: subroutine galacticus_function_classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; file merger_trees.construct.read.F90; file merger_trees.operators.assign_orbits.F90; module node_component_satellite_orbiting; module node_component_satellite_standard; module node_component_satellite_very_simple; module node_component_spin_vitvitska

file: `satellites.merging.virial_orbits.Jiang2014.F90`
Description: An implementation of virial orbits using the Jiang et al. [2014] orbital parameter distribution.
Code lines: 547
Modules used: cosmology_functions cosmology_parameters dark_matter_halo_scales statistics_distributions

tables

function: jiang2014constructorinternal
Description: Internal constructor for the jiang2014 virial orbits class.
Code lines: 134
Contained by: file satellites.merging.virial_orbits.Jiang2014.F90
Modules used: fgsl numerical_integration

function: jiang2014distributionvelocitytangential
Description: The integrand used to average the tangential velocity over the distribution function for total velocity.
Code lines: 10
Contained by: function jiang2014constructorinternal
Modules used: bessel_functions numerical_constants_math struve_functions

function: jiang2014distributionvelocitytotalsquared
Description: The integrand used to average the squared total velocity over the distribution function for total velocity.
Code lines: 7
Contained by: function jiang2014constructorinternal

function: jiang2014constructorparameters
Description: Generic constructor for the jiang2014 virial orbits class.
Code lines: 116
Contained by: file satellites.merging.virial_orbits.Jiang2014.F90
Modules used: input_parameters

function: jiang2014densitycontrastdefinition
Description: Return a virial density contrast object defining that used in the definition of Jiang et al. [2014] virial orbits.
Code lines: 8
Contained by: file satellites.merging.virial_orbits.Jiang2014.F90

subroutine: jiang2014destructor
Description: Destructor for the jiang2014 virial orbits class.
Code lines: 10
Contained by: file satellites.merging.virial_orbits.Jiang2014.F90

function: jiang2014orbit
Description: Return jiang2014 orbital parameters for a satellite.
Code lines: 89
Contained by: file satellites.merging.virial_orbits.Jiang2014.F90
Modules used: dark_matter_profile_mass_definitions galacticus_error galacticus_nodes root_finder

subroutine: jiang2014parametersselect
Description: Select the parameter set to use for this satellite/host pairing.
Code lines: 23

Contained by: file satellites.merging.virial_orbits.Jiang2014.F90

function: jiang2014radialvelocitycdf
Description: Cumulative distribution function for the radial velocity.
Code lines: 7
Contained by: file satellites.merging.virial_orbits.Jiang2014.F90

function: jiang2014totalvelocitycdf
Description: Cumulative distribution function for the total velocity.
Code lines: 7
Contained by: file satellites.merging.virial_orbits.Jiang2014.F90

function: jiang2014velocitytangentialmagnitudemean
Description: Return the mean magnitude of the tangential velocity.
Code lines: 21
Contained by: file satellites.merging.virial_orbits.Jiang2014.F90
Modules used: dark_matter_profile_mass_definitions galacticus_nodes

function: jiang2014velocitytangentialvectormean
Description: Return the mean of the vector tangential velocity.
Code lines: 12
Contained by: file satellites.merging.virial_orbits.Jiang2014.F90
Modules used: galacticus_error

function: jiang2014velocitytotalrootmeansquared
Description: Return the root mean squared total velocity.
Code lines: 21
Contained by: file satellites.merging.virial_orbits.Jiang2014.F90
Modules used: dark_matter_profile_mass_definitions galacticus_nodes

interface: virialorbitjiang2014
Description: Constructors for the jiang2014 virial orbit class.
Code lines: 4
Contained by: file satellites.merging.virial_orbits.Jiang2014.F90

file: satellites.merging.virial_orbits.Wetzel2010.F90
Description: An implementation of virial orbits using the Wetzel [2010] orbital parameter distribution.
Code lines: 299
Modules used: cosmological_density_field cosmology_functions
dark_matter_halo_scales root_finder
tables

interface: virialorbitwetzel2010
Description: Constructors for the wetzel2010 virial orbit class.
Code lines: 4
Contained by: file satellites.merging.virial_orbits.Wetzel2010.F90

function: wetzel2010circularitycumulativeprobability
Description: The cumulative probability distribution for orbital circularity.

Code lines: 8
Contained by: file satellites.merging.virial_orbits.Wetzel2010.F90
Modules used: hypergeometric_functions

function: wetzel2010circularityroot
Description: Function used in finding the circularity corresponding to a given cumulative probability.
Code lines: 8
Contained by: file satellites.merging.virial_orbits.Wetzel2010.F90

function: wetzel2010constructorinternal
Description: Internal constructor for the wetzel2010 virial orbits class.
Code lines: 36
Contained by: file satellites.merging.virial_orbits.Wetzel2010.F90
Modules used: hypergeometric_functions

function: wetzel2010constructorparameters
Description: Constructor for the wetzel2010 virial orbits class which takes a parameter set as input.
Code lines: 19
Contained by: file satellites.merging.virial_orbits.Wetzel2010.F90
Modules used: input_parameters

function: wetzel2010densitycontrastdefinition
Description: Return a virial density contrast object defining that used in the definition of Wetzel [2010] virial orbits.
Code lines: 8
Contained by: file satellites.merging.virial_orbits.Wetzel2010.F90

subroutine: wetzel2010destructor
Description: Destructor for the wetzel2010 virial orbits class.
Code lines: 10
Contained by: file satellites.merging.virial_orbits.Wetzel2010.F90

function: wetzel2010orbit
Description: Return wetzel2010 orbital parameters for a satellite.
Code lines: 80
Contained by: file satellites.merging.virial_orbits.Wetzel2010.F90
Modules used: dark_matter_profile_mass_definitions galacticus_error galacticus_nodes

function: wetzel2010velocitytangentialmagnitudemean
Description: Return the mean magnitude of the tangential velocity.
Code lines: 11
Contained by: file satellites.merging.virial_orbits.Wetzel2010.F90
Modules used: galacticus_error

function: wetzel2010velocitytangentialvectormean
Description: Return the mean of the vector tangential velocity.
Code lines: 12
Contained by: file satellites.merging.virial_orbits.Wetzel2010.F90

Modules used: galacticus_error

function: wetzel2010velocitytotalrootmeansquared
Description: Return the root mean squared total velocity.
Code lines: 11
Contained by: file satellites.merging.virial_orbits.Wetzel2010.F90
Modules used: galacticus_error

file: satellites.merging.virial_orbits.fixed.F90
Description: An implementation of virial orbits which assumes fixed orbital parameters.
Code lines: 223
Modules used: dark_matter_halo_scales

function: fixedconstructorinternal
Description: Internal constructor for the fixed virial orbits class.
Code lines: 11
Contained by: file satellites.merging.virial_orbits.fixed.F90
Modules used: galacticus_error

function: fixedconstructorparameters
Description: Constructor for the fixed satellite virial orbit class which takes a parameter set as input.
Code lines: 33
Contained by: file satellites.merging.virial_orbits.fixed.F90
Modules used: input_parameters

function: fixeddensitycontrastdefinition
Description: Return a virial density contrast object defining that used in the definition of fixed virial orbits.
Code lines: 8
Contained by: file satellites.merging.virial_orbits.fixed.F90

subroutine: fixeddestructor
Description: Destructor for the fixed virial orbits class.
Code lines: 8
Contained by: file satellites.merging.virial_orbits.fixed.F90

function: fixedorbit
Description: Return fixed orbital parameters for a satellite.
Code lines: 57
Contained by: file satellites.merging.virial_orbits.fixed.F90
Modules used: dark_matter_profile_mass_definitions galacticus_display galacticus_error galacticus_nodes iso_varying_string

function: fixedvelocitytangentialmagnitudemean
Description: Return the mean magnitude of the tangential velocity.
Code lines: 15
Contained by: file satellites.merging.virial_orbits.fixed.F90
Modules used: dark_matter_profile_mass_definitions galacticus_nodes

function: `fixedvelocitytangentialvectormean`
Description: Return the mean of the vector tangential velocity.
Code lines: 12
Contained by: file satellites.merging.virial_orbits.fixed.F90
Modules used: galacticus_error

function: `fixedvelocitytotalrootmeansquared`
Description: Return the root mean squared of the total velocity.
Code lines: 15
Contained by: file satellites.merging.virial_orbits.fixed.F90
Modules used: dark_matter_profile_mass_definitions galacticus_nodes

interface: `virialorbitfixed`
Description: Constructors for the `fixed` virial orbit class.
Code lines: 4
Contained by: file satellites.merging.virial_orbits.fixed.F90

file: `satellites.merging.virial_orbits.isotropic.F90`
Description: An implementation of virial orbits which assumes an isotropic distribution of infall directions and tangential velocities applied to another virial orbit class.
Code lines: 142

function: `isotropicconstructorinternal`
Description: Internal constructor for the `isotropic` virial orbits class.
Code lines: 8
Contained by: file satellites.merging.virial_orbits.isotropic.F90

function: `isotropicconstructorparameters`
Description: Constructor for the `isotropic` satellite virial orbit class which takes a parameter set as input.
Code lines: 13
Contained by: file satellites.merging.virial_orbits.isotropic.F90
Modules used: input_parameters

function: `isotropicdensitycontrastdefinition`
Description: Return a virial density contrast object defining that used in the definition of virial orbits.
Code lines: 8
Contained by: file satellites.merging.virial_orbits.isotropic.F90

subroutine: `isotropicdestructor`
Description: Destructor for the `isotropic` virial orbits class.
Code lines: 7
Contained by: file satellites.merging.virial_orbits.isotropic.F90

function: `isotropicorbit`
Description: Return isotropic orbital parameters for a satellite.
Code lines: 16
Contained by: file satellites.merging.virial_orbits.isotropic.F90

Modules used: numerical_constants_math

function: isotropicvelocitytangentialmagnitudemean
Description: Return the mean magnitude of the tangential velocity.
Code lines: 8
Contained by: file satellites.merging.virial_orbits.isotropic.F90

function: isotropicvelocitytangentialvectormean
Description: Return the mean of the vector tangential velocity.
Code lines: 12
Contained by: file satellites.merging.virial_orbits.isotropic.F90
Modules used: galacticus_error

function: isotropicvelocitytotalrootmeansquared
Description: Return the root mean squared of the total velocity.
Code lines: 8
Contained by: file satellites.merging.virial_orbits.isotropic.F90

interface: virialorbitisotropic
Description: Constructors for the isotropic virial orbit class.
Code lines: 4
Contained by: file satellites.merging.virial_orbits.isotropic.F90

file: satellites.merging.virial_orbits.spin_correlated.F90
Description: An implementation of virial orbits which assumes that the orbital angular momentum is correlated with the spin vector of the host halo.
Code lines: 189

function: spincorrelatedconstructorinternal
Description: Internal constructor for the spinCorrelated virial orbits class.
Code lines: 9
Contained by: file satellites.merging.virial_orbits.spin_correlated.F90

function: spincorrelatedconstructorparameters
Description: Constructor for the spinCorrelated satellite virial orbit class which takes a parameter set as input.
Code lines: 22
Contained by: file satellites.merging.virial_orbits.spin_correlated.F90
Modules used: input_parameters

function: spincorrelateddensitycontrastdefinition
Description: Return a virial density contrast object defining that used in the definition of virial orbits.
Code lines: 8
Contained by: file satellites.merging.virial_orbits.spin_correlated.F90

subroutine: spincorrelateddestructor
Description: Destructor for the spinCorrelated virial orbits class.
Code lines: 7
Contained by: file satellites.merging.virial_orbits.spin_correlated.F90

function: `spincorrelatedorbit`
Description: Return spinCorrelated orbital parameters for a satellite.
Code lines: 53
Contained by: file satellites.merging.virial_orbits.spin_correlated.F90
Modules used: coordinates, galacticus_error, galacticus_nodes, numerical_constants_math, vectors

function: `spincorrelatedvelocitytangentialmagnitudemean`
Description: Return the mean magnitude of the tangential velocity.
Code lines: 8
Contained by: file satellites.merging.virial_orbits.spin_correlated.F90

function: `spincorrelatedvelocitytangentialvectormean`
Description: Return the mean of the vector tangential velocity.
Code lines: 12
Contained by: file satellites.merging.virial_orbits.spin_correlated.F90
Modules used: galacticus_nodes

function: `spincorrelatedvelocitytotalrootmeansquared`
Description: Return the root mean squared of the total velocity.
Code lines: 8
Contained by: file satellites.merging.virial_orbits.spin_correlated.F90

interface: `virialorbitspincorrelated`
Description: Constructors for the `spinCorrelated` virial orbit class.
Code lines: 4
Contained by: file satellites.merging.virial_orbits.spin_correlated.F90

file: `satellites.orbits.F90`
Description: Contains a module which implements calculations related to satellite orbits.
Code lines: 292

module: `satellite_orbits`
Description: Implements calculations related to satellite orbits.
Code lines: 270
Contained by: file satellites.orbits.F90
Modules used: dark_matter_profiles_dmo, galacticus_nodes, kind_numbers
Used by: function font2008force; function satelliteorbitalextremaextract; subroutine node_component_dynamics_-statistics_bars_record; function boylankolchin2008timeuntilmerging; function jiang2008timeuntilmerging; function randomtimeuntilmerging; function sphericalsymmetrytidaltensorradial

function: `equivalent_circular_orbit_solver`
Description: Root function used in finding equivalent circular orbits.

Code lines: 25
Contained by: module satellite_orbits
Modules used: galactic_structure_options, galactic_structure_potentials, galacticus_error

function: `extremum_solver`
Description: Root function used in finding orbital extremum radius.
Code lines: 10
Contained by: module satellite_orbits
Modules used: galactic_structure_potentials

function: `satellite_orbit_convert_to_current_potential`
Description: Takes a virial orbit and adjusts the energy to account for the change in the definition of potential between the original halo in which the orbit was defined and the current halo. Since the potential at the virial radius of halos is always defined to be $\Phi(r_{\rm vir}) = -V_{\rm vir}^2$ then the specific energy transforms as:

$$e \rightarrow e + V_{\rm vir,0}^2 + \Phi(r_{\rm vir,0}), \tag{18.40}$$

where subscript 0 refers to the original halo in which the orbit was defined and $\Phi(r)$ is the potential of the current halo.
Code lines: 26
Contained by: module satellite_orbits
Modules used: galactic_structure_potentials, kepler_orbits, numerical_constants_physical

function: `satellite_orbit_equivalent_circular_orbit_radius`
Description: Solves for the equivalent circular orbit radius for `orbit` in `nodeHost`.
Code lines: 40
Contained by: module satellite_orbits
Modules used: dark_matter_halo_scales, kepler_orbits, root_finder

subroutine: `satellite_orbit_extremum_phase_space_coordinates`
Description: Solves for the pericentric radius and velocity of `orbit` in `nodeHost`.
Code lines: 107
Contained by: module satellite_orbits
Modules used: galactic_structure_options, galactic_structure_potentials, galacticus_error, galacticus_nodes, kepler_orbits, numerical_constants_physical, numerical_constants_prefixes, root_finder

subroutine: `satellite_orbit_reset`
Description: Reset the satellite orbit calculations.
Code lines: 9
Contained by: module satellite_orbits

file: `satellites.orphans.distributions.F90`
Description: Contains a module that provides a class for distributions of orphan satellites.

Code lines: 52

module: `satellite_oprhan_distributions`

Description: Provides a class for dark matter halo spin distributions.
Code lines: 30
Contained by: file satellites.orphans.distributions.F90
Modules used: galacticus_nodes
Used by: subroutine galacticus_function_classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; file merger_trees.operators.prune_lightcone.F90; module node_component_position_preset_orphans; subroutine node_component_position_trace_dark_matter_assign

file: `satellites.orphans.distributions.random_isotropic.F90`

Description: An abstract implementation of the orphan satellite distribution which assumes an isotropic distribution with randomly assigned positions.
Code lines: 131
Modules used: statistics_distributions

function: `randomisotropicposition`

Description: Return the position of an orphan satellite in a random isotropic distribution.
Code lines: 24
Contained by: file satellites.orphans.distributions.random_isotropic.F90
Modules used: coordinates; galacticus_nodes; numerical_constants_math; pseudo_random

function: `randomisotropicvelocity`

Description: Return the velocity of an orphan satellite in a random isotropic distribution.
Code lines: 29
Contained by: file satellites.orphans.distributions.random_isotropic.F90
Modules used: coordinates; galacticus_nodes; numerical_constants_math; pseudo_random

type: `satelliteorphandistributionrandomisotropic`

Description: An abstract orphan satellite distribution which assumes an isotropic, random distribution.
Code lines: 23
Contained by: file satellites.orphans.distributions.random_isotropic.F90

file: `satellites.orphans.distributions.trace_dark_matter.F90`

Description: An abstract implementation of the orphan satellite distribution which assumes an isotropic distribution with randomly assigned positions.
Code lines: 121
Modules used: dark_matter_halo_scales

interface: `satelliteorphandistributiontracedarkmatter`

Description: Constructors for the `traceDarkMatter` orphan satellite distribution class.
Code lines: 4
Contained by: file satellites.orphans.distributions.trace_dark_matter.F90

function: `tracedarkmatterconstructorinternal`
Description: Internal constructor for the `traceDarkMatter` orphan satellite distribution class.
Code lines: 8
Contained by: file satellites.orphans.distributions.trace_dark_matter.F90

function: `tracedarkmatterconstructorparameters`
Description: Constructor for the `traceDarkMatter` orphan satellite distribution class which takes a parameter list as input.
Code lines: 15
Contained by: file satellites.orphans.distributions.trace_dark_matter.F90
Modules used: input_parameters

subroutine: `tracedarkmatterdestructor`
Description: Destructor for the `traceDarkMatter` orphan satellite distribution class.
Code lines: 7
Contained by: file satellites.orphans.distributions.trace_dark_matter.F90

function: `tracedarkmatterextent`
Description: Return the maximum extent of the orphan satellite population for the `traceDarkMatter` orphan satellite distribution class.
Code lines: 10
Contained by: file satellites.orphans.distributions.trace_dark_matter.F90

function: `tracedarkmatterinversecmfradial`
Description: Return the radial coordinate within which the given `fraction` of orphan satellites are found.
Code lines: 14
Contained by: file satellites.orphans.distributions.trace_dark_matter.F90
Modules used: galactic_structure_enclosed_masses galactic_structure_options

function: `tracedarkmattervelocitydispersion`
Description: Return the velocity dispersion of the orphan satellite population.
Code lines: 10
Contained by: file satellites.orphans.distributions.trace_dark_matter.F90

file: `satellites.promotion.F90`
Description: Contains a module which handles events where a satellite is moved to a new host halo.
Code lines: 104

module: `satellite_promotion`
Description: Handles events where a satellite is moved to a new host halo.
Code lines: 82
Contained by: file satellites.promotion.F90
Used by: subroutine singlelevelhierarchyprocess

subroutine: `satellite_move_to_new_host`
Description: Move `satelliteNode` to be a satellite of `newHostNode`.
Code lines: 72
Contained by: module satellite_promotion

Modules used: galacticus_display, galacticus_nodes, iso_varying_string, node_component_position_preset, node_component_position_trace_dark_matter, node_component_satellite_preset, node_component_satellite_standard, node_component_satellite_very_simple, string_handling

file: satellites.tidal_fields.F90
Description: Contains a module that implements calculations of tidal fields acting on satellites.
Code lines: 40

module: satellites_tidal_fields
Description: Implements calculations of tidal fields acting on satellites.
Code lines: 18
Contained by: file satellites.tidal_fields.F90
Modules used: galacticus_nodes
Used by: file galactic_dynamics.bar_instability.Efstathiou1982Tidal.F90, subroutine galacticus_function_classes_destroy, subroutine galacticus_state_retrieve, subroutine galacticus_state_store, module node_component_spheroid_standard, file tidal_stripping.mass_loss_rate.disks.simple.F90, file tidal_stripping.mass_loss_rate.spheroids.simple.F90

file: satellites.tidal_fields.null.F90
Description: Implements a satellite tidal field class which assumes zero tidal field.
Code lines: 60

function: nullconstructorparameters
Description: Constructor for the null satellite tidal field class which builds the object from a parameter set.
Code lines: 10
Contained by: file satellites.tidal_fields.null.F90
Modules used: input_parameters

function: nulltidaltensorradial
Description: Return the radial part of the tidal tensor for satellite halos assumed to be zero.
Code lines: 9
Contained by: file satellites.tidal_fields.null.F90

interface: satellitetidalfieldnull
Description: Constructors for the null satellite tidal field class.
Code lines: 3
Contained by: file satellites.tidal_fields.null.F90

file: satellites.tidal_fields.spherical_symmetry.F90
Description: Contains a module which implements a model of the tidal field acting on a satellite assuming spherical symmetry in the host.
Code lines: 111

interface: `satellitetidalfieldsphericalsymmetry`
Description: Constructors for the sphericalSymmetry satellite tidal field class.
Code lines: 4
Contained by: file `satellites.tidal_fields.spherical_symmetry.F90`

function: `sphericalsymmetryconstructorinternal`
Description: Internal constructor for the `sphericalSymmetry` satellite tidal field class.
Code lines: 8
Contained by: file `satellites.tidal_fields.spherical_symmetry.F90`

function: `sphericalsymmetryconstructorparameters`
Description: Constructor for the `sphericalSymmetry` satellite tidal field class which builds the object from a parameter set.
Code lines: 19
Contained by: file `satellites.tidal_fields.spherical_symmetry.F90`
Modules used: `input_parameters`

function: `sphericalsymmetrytidaltensorradial`
Description: Return the radial part of the tidal tensor for satellite halos assuming spherical symmetry of the host.
Code lines: 39
Contained by: file `satellites.tidal_fields.spherical_symmetry.F90`
Modules used:

`galactic_structure_densities`	`galactic_structure_enclosed_masses`
`galactic_structure_options`	`galacticus_nodes`
`kepler_orbits`	`numerical_constants_math`
`numerical_constants_physical`	`satellite_orbits`

file: `satellites.tidal_heating.rate.F90`
Description: Contains a module that implements a class for computing tidal heating rates for satellites.
Code lines: 42

module: `satellite_tidal_heating`
Description: Implements a class for calculations of tidal heating for satellites.
Code lines: 18
Contained by: file `satellites.tidal_heating.rate.F90`
Modules used: `galacticus_nodes`
Used by:

subroutine `galacticus_function_-classes_destroy`	subroutine `galacticus_state_retrieve`
subroutine `galacticus_state_store`	module `node_component_satellite_-orbiting`

file: `satellites.tidal_heating.rate.Gnedin1999.F90`
Description: Contains a class which implements the tidal heating rate model of Gnedin et al. [1999].
Code lines: 169
Modules used:

`cosmology_parameters`	`dark_matter_halo_scales`

function: `gnedin1999constructorinternal`
Description: Internal constructor for the `gnedin1999` satellite tidal heating rate class.

Code lines: 10
Contained by: file satellites.tidal_heating.rate.Gnedin1999.F90

function: gnedin1999constructorparameters
Description: Constructor for the gnedin1999 satellite tidal heating rate class which builds the object from a parameter set.
Code lines: 35
Contained by: file satellites.tidal_heating.rate.Gnedin1999.F90
Modules used: input_parameters

subroutine: gnedin1999destructor
Description: Default constructor for the gnedin1999 satellite tidal heating rate class.
Code lines: 8
Contained by: file satellites.tidal_heating.rate.Gnedin1999.F90

function: gnedin1999heatingrate
Description: Return the tidal heating rate for satellite halos assuming the model of Gnedin et al. [1999].
Code lines: 59
Contained by: file satellites.tidal_heating.rate.Gnedin1999.F90
Modules used: error_functions, galactic_structure_densities, galactic_structure_enclosed_masses, galactic_structure_options, galactic_structure_rotation_curves, galacticus_nodes, numerical_constants_astronomical, numerical_constants_math, numerical_constants_physical, numerical_constants_prefixes, tensors, vectors

interface: satellitetidalheatingrategnedin1999
Description: Constructors for the gnedin1999 satellite tidal heating rate class.
Code lines: 4
Contained by: file satellites.tidal_heating.rate.Gnedin1999.F90

file: satellites.tidal_heating.rate.zero.F90
Description: Contains a class which implements a tidal heating rate model in which the heating rate is always zero.
Code lines: 62

interface: satellitetidalheatingratezero
Description: Constructors for the zero satellite tidal heating rate class.
Code lines: 3
Contained by: file satellites.tidal_heating.rate.zero.F90

function: zeroconstructorparameters
Description: Constructor for the zero satellite tidal heating rate class which builds the object from a parameter set.
Code lines: 10
Contained by: file satellites.tidal_heating.rate.zero.F90
Modules used: input_parameters

function: zeroheatingrate

Description: Return the the tidal heating rate for the given node.
Code lines: 9
Contained by: file satellites.tidal_heating.rate.zero.F90

file: satellites.tidal_stripping.rate.F90
Description: Contains a module that provides a class to perform calculations of the mass loss rate due to tidal stripping for satellites.
Code lines: 43

module: satellite_tidal_stripping
Description: Provides a class to perform calculations of the mass loss rate due to tidal stripping for satellites.
Code lines: 19
Contained by: file satellites.tidal_stripping.rate.F90
Modules used: galacticus_nodes
Used by: subroutine galacticus_function_classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; module node_component_satellite_orbiting

file: satellites.tidal_stripping.rate.Zentner2005.F90
Description: Implementation of a satellite tidal stripping class which follows the model of Zentner et al. [2005].
Code lines: 234
Modules used: dark_matter_halo_scales; kind_numbers

interface: satellitetidalstrippingzentner2005
Description: Constructors for the zentner2005 satellite tidal stripping class.
Code lines: 4
Contained by: file satellites.tidal_stripping.rate.Zentner2005.F90

subroutine: zentner2005autohook
Description: Attach to the calculation reset event.
Code lines: 8
Contained by: file satellites.tidal_stripping.rate.Zentner2005.F90
Modules used: events_hooks

subroutine: zentner2005calculationreset
Description: Reset the stored tidal radii.
Code lines: 9
Contained by: file satellites.tidal_stripping.rate.Zentner2005.F90

function: zentner2005constructorinternal
Description: Internal constructor for the zentner2005 satellite tidal stripping class.
Code lines: 10
Contained by: file satellites.tidal_stripping.rate.Zentner2005.F90

function: zentner2005constructorparameters
Description: Constructor for the zentner2005 satellite tidaql stripping class which builds the object from a parameter set.
Code lines: 23

Contained by: file satellites.tidal_stripping.rate.Zentner2005.F90
Modules used: input_parameters

subroutine: zentner2005destructor
Description: Destructor for the zentner2005 satellite tidal stripping class.
Code lines: 9
Contained by: file satellites.tidal_stripping.rate.Zentner2005.F90
Modules used: events_hooks

function: zentner2005masslossrate
Description: Return a mass loss rate for satellites due to tidal stripping using the formulation of Zentner et al. [2005].
Code lines: 84
Contained by: file satellites.tidal_stripping.rate.Zentner2005.F90
Modules used: dark_matter_halo_scales, error_functions, galactic_structure_densities, galactic_structure_enclosed_masses, galactic_structure_options, galacticus_nodes, numerical_constants_astronomical, numerical_constants_math, numerical_constants_physical, numerical_constants_prefixes, root_finder, vectors

function: zentner2005tidalradiussolver
Description: Root function used to find the tidal radius within a subhalo.
Code lines: 14
Contained by: file satellites.tidal_stripping.rate.Zentner2005.F90
Modules used: galactic_structure_enclosed_masses, numerical_constants_astronomical, numerical_constants_physical, numerical_constants_prefixes

file: satellites.tidal_stripping.rate.zero.F90
Description: Implementation of a satellite tidal stripping class in which the stripping rate is always zero.
Code lines: 65
Modules used: dark_matter_halo_scales

interface: satellitetidalstrippingzero
Description: Constructors for the zero satellite tidal stripping class.
Code lines: 3
Contained by: file satellites.tidal_stripping.rate.zero.F90

function: zeroconstructorparameters
Description: Constructor for the zero satellite tidal stripping class which builds the object from a parameter set.
Code lines: 10
Contained by: file satellites.tidal_stripping.rate.zero.F90
Modules used: input_parameters

function: zeromasslossrate
Description: Return a mass loss rate for satellites due to tidal stripping which is always zero.
Code lines: 9
Contained by: file satellites.tidal_stripping.rate.zero.F90

file: `shocks.one_dimensional.F90`
Description: Contains a module which implements calculations of one-dimensional shocks.
Code lines: 45

module: `shocks_1d`
Description: iImplements calculations of one-dimensional shocks.
Code lines: 23
Contained by: file `shocks.one_dimensional.F90`
Used by: function `coldmodecoldmodefraction`

function: `shocks_1d_density_jump`
Description: Computes the density jump across a one-dimensional shock.
Code lines: 11
Contained by: module `shocks_1d`

file: `star_formation.feedback.disks.Creasey2012.F90`
Description: Implementation of the Creasey et al. [2012] outflow rate due to star formation feedback in galactic disks.
Code lines: 177
Modules used: `star_formation_rate_surface_density_-disks`

function: `creasey2012constructorinternal`
Description: Internal constructor for the `creasey2012` star formation feedback from disks class.
Code lines: 9
Contained by: file `star_formation.feedback.disks.Creasey2012.F90`

function: `creasey2012constructorparameters`
Description: Constructor for the Creasey et al. [2012] star formation feedback in disks class which takes a parameter set as input.
Code lines: 41
Contained by: file `star_formation.feedback.disks.Creasey2012.F90`
Modules used: `galacticus_error`

subroutine: `creasey2012destructor`
Description: Destructor for the `creasey2012` feedback in disks class.
Code lines: 7
Contained by: file `star_formation.feedback.disks.Creasey2012.F90`

function: `creasey2012outflowrate`

Description: Returns the outflow rate (in $M_\odot$ Gyr^{-1}) for star formation in the galactic disk of `thisNode` using the model of Creasey et al. [2012]. The outflow rate is given by

$$\dot{M}_\mathrm{outflow} = \int_0^\infty \beta_0 \Sigma_{g,1}^{-\mu}(r) f_\mathrm{g}^\nu(r) \dot{\Sigma}_\star(r) 2\pi r \mathrm{d}r, \tag{18.41}$$

where $\Sigma_{g,1}(r)$ is the surface density of gas in units of $M_\odot$ pc^{-2}, $f_\mathrm{g}(r)$ is the gas fraction, $\dot{\Sigma}_\star(r)$ is the surface density of star formation rate, $\beta_0 =$`[beta0]`, $\mu =$`[mu]`, and $\nu =$`[nu]`.

Code lines: 68

Contained by: file `star_formation.feedback.disks.Creasey2012.F90`

Modules used: `fgsl` `galacticus_nodes` `numerical_constants_math` `numerical_integration` `stellar_feedback`

function: `outflowrateintegrand`

Description: Integrand function for the "Creasey et al. (2012)" supernovae feedback calculation.
Code lines: 22
Contained by: function creasey2012outflowrate
Modules used: galactic_structure_options galactic_structure_surface_densities numerical_constants_prefixes

interface: starformationfeedbackdiskscreasey2012
Description: Constructors for the creasey2012 fraction star formation feedback in disks class.
Code lines: 4
Contained by: file star_formation.feedback.disks.Creasey2012.F90

file: star_formation.feedback.disks.F90
Description: Contains a module which provides an object that implements feedback from star formation in disks.
Code lines: 41

module: star_formation_feedback_disks
Description: Provides an object that implements calculations of feedback from star formation in disks.
Code lines: 19
Contained by: file star_formation.feedback.disks.F90
Modules used: galacticus_nodes
Used by: subroutine galacticus_function_classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; module node_component_disk_standard; module node_component_disk_very_simple; subroutine node_component_disk_very_simple_rate_compute

file: star_formation.feedback.disks.fixed.F90
Description: Implementation of a fixed fraction outflow rate due to star formation feedback in galactic disks.
Code lines: 85

function: fixedconstructorinternal
Description: Internal constructor for the fixed star formation feedback from disks class.
Code lines: 8
Contained by: file star_formation.feedback.disks.fixed.F90

function: fixedconstructorparameters
Description: Constructor for the fixed fraction star formation feedback in disks class which takes a parameter set as input.
Code lines: 20
Contained by: file star_formation.feedback.disks.fixed.F90
Modules used: galacticus_error input_parameters

function: fixedoutflowrate
Description: Returns the outflow rate (in $M_\odot$ Gyr^{-1}) for star formation in the galactic disk of node. Assumes a fixed ratio of outflow rate to star formation rate.
Code lines: 12
Contained by: file star_formation.feedback.disks.fixed.F90
Modules used: stellar_feedback

interface: `starformationfeedbackdisksfixed`
Description: Constructors for the fixed fraction star formation feedback in disks class.
Code lines: 4
Contained by: file `star_formation.feedback.disks.fixed.F90`

file: `star_formation.feedback.disks.halo_scaling.F90`
Description: Implementation of an outflow rate due to star formation feedback in galactic disks which scales with halo velocity.
Code lines: 150
Modules used: `cosmology_functions` `dark_matter_halo_scales`

function: `haloscalingconstructorinternal`
Description: Internal constructor for the halo scaling star formation feedback from disks class.
Code lines: 19
Contained by: file `star_formation.feedback.disks.halo_scaling.F90`
Modules used: `stellar_feedback`

function: `haloscalingconstructorparameters`
Description: Constructor for the halo scaling fraction star formation feedback in disks class which takes a parameter set as input.
Code lines: 41
Contained by: file `star_formation.feedback.disks.halo_scaling.F90`
Modules used: `galacticus_error`

subroutine: `haloscalingdestructor`
Description: Destructor for the halo scaling feedback from star formation in disks class.
Code lines: 8
Contained by: file `star_formation.feedback.disks.halo_scaling.F90`

function: `haloscalingoutflowrate`
Description: Returns the outflow rate (in $M_\odot$ Gyr^{-1}) for star formation in the galactic disk of `node`.
Code lines: 29
Contained by: file `star_formation.feedback.disks.halo_scaling.F90`
Modules used: `galacticus_nodes`

interface: `starformationfeedbackdiskshaloscaling`
Description: Constructors for the halo scaling fraction star formation feedback in disks class.
Code lines: 4
Contained by: file `star_formation.feedback.disks.halo_scaling.F90`

file: `star_formation.feedback.disks.power_law.F90`
Description: Implementation of a power-law outflow rate due to star formation feedback in galactic disks.
Code lines: 136

function: `powerlawconstructorinternal`
Description: Internal constructor for the power-law star formation feedback from disks class.
Code lines: 14
Contained by: file `star_formation.feedback.disks.power_law.F90`

Modules used: galacticus_error

function: powerlawconstructorparameters
Description: Constructor for the power-law star formation feedback in disks class which takes a parameter set as input.
Code lines: 26
Contained by: file star_formation.feedback.disks.power_law.F90

function: powerlawoutflowrate
Description: Returns the outflow rate (in $M_\odot$ Gyr^{-1}) for star formation in the galactic disk of thisNode. The outflow rate is given by

$$\dot{M}_\mathrm{outflow} = \left(\frac{V_\mathrm{disk,outflow}}{V_\mathrm{disk}} \right)^{\alpha_\mathrm{disk,outflow}}, \tag{18.42}$$

where $V_\mathrm{disk,outflow}$(=diskOutflowVelocity) is the velocity scale at which outflow rate equals star formation rate and $\alpha_{disk,outflow}$(=diskOutflowExponent) controls the scaling with velocity. Note that the velocity V_disk is whatever characteristic value returned by the disk method. This scaling is functionally similar to that adopted by Cole et al. [2000], except that they specifically used the circular velocity at half-mass radius.
Code lines: 29
Contained by: file star_formation.feedback.disks.power_law.F90
Modules used: galacticus_nodes stellar_feedback

function: powerlawvelocitycharacteristic
Description: Return the characteristic velocity for power-law feedback models in disks. In this case the characteristic velocity is a constant.
Code lines: 10
Contained by: file star_formation.feedback.disks.power_law.F90

interface: starformationfeedbackdiskspowerlaw
Description: Constructors for the power-law star formation feedback in disks class.
Code lines: 4
Contained by: file star_formation.feedback.disks.power_law.F90

file: star_formation.feedback.disks.power_law.redshift_scaling.F90
Description: Implementation of a power-law outflow rate due to star formation feedback in galactic disks in which the characteristic velocity scales as a power of $(1+z)$.
Code lines: 100
Modules used: cosmology_functions

function: powerlawredshiftscalingconstructorinternal
Description: Internal constructor for the power-law redshift-scaling star formation feedback from disks class.
Code lines: 10
Contained by: file star_formation.feedback.disks.power_law.redshift_scaling.F90

function: powerlawredshiftscalingconstructorparameters
Description: Constructor for the power-law redshift-scaling star formation feedback in disks class which takes a parameter set as input.
Code lines: 20

Contained by: file star_formation.feedback.disks.power_law.redshift_scaling.F90

subroutine: powerlawredshiftscalingdestructor
Description: Destructor for the power-law redshift-scaling feedback from star formation in disks class.
Code lines: 7
Contained by: file star_formation.feedback.disks.power_law.redshift_scaling.F90

function: powerlawredshiftscalingvelocitycharacteristic
Description: Return the characteristic velocity for power-law feedback models in disks. In this case the characteristic velocity is a constant.
Code lines: 12
Contained by: file star_formation.feedback.disks.power_law.redshift_scaling.F90
Modules used: galacticus_nodes

interface: starformationfeedbackdiskspowerlawredshiftscaling
Description: Constructors for the power-law redshift-scaling star formation feedback in disks class.
Code lines: 4
Contained by: file star_formation.feedback.disks.power_law.redshift_scaling.F90

file: star_formation.feedback.disks.velocity_maximum_scaling.F90
Description: Implementation of an outflow rate due to star formation feedback in galactic disks which scales with peak halo velocity.
Code lines: 156
Modules used: cosmology_functions dark_matter_profiles_dmo math_exponentiation

interface: starformationfeedbackdisksvlctymxsclng
Description: Constructors for the velocity maximum scaling fraction star formation feedback in disks class.
Code lines: 4
Contained by: file star_formation.feedback.disks.velocity_maximum_scaling.F90

function: vlctymxsclngconstructorinternal
Description: Internal constructor for the halo scaling star formation feedback from disks class.
Code lines: 22
Contained by: file star_formation.feedback.disks.velocity_maximum_scaling.F90
Modules used: stellar_feedback

function: vlctymxsclngconstructorparameters
Description: Constructor for the velocity maximum scaling fraction star formation feedback in disks class which takes a parameter set as input.
Code lines: 42
Contained by: file star_formation.feedback.disks.velocity_maximum_scaling.F90
Modules used: galacticus_error

subroutine: vlctymxsclngdestructor
Description: Destructor for the velocity maximum scaling feedback from star formation in disks class.
Code lines: 8
Contained by: file star_formation.feedback.disks.velocity_maximum_scaling.F90

function: `vlctymxsclngoutflowrate`
Description: Returns the outflow rate (in $M_\odot$ Gyr^{-1}) for star formation in the galactic disk of `node`.
Code lines: 29
Contained by: file star_formation.feedback.disks.velocity_maximum_scaling.F90
Modules used: galacticus_nodes

file: `star_formation.feedback.spheroids.F90`
Description: Contains a module which provides an object that implements cosmological parameters.
Code lines: 41

module: `star_formation_feedback_spheroids`
Description: Provides an object that implements calculations of feedback from star formation in spheroids.
Code lines: 19
Contained by: file star_formation.feedback.spheroids.F90
Modules used: galacticus_nodes
Used by: subroutine galacticus_function_classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; module node_component_spheroid_standard; module node_component_spheroid_very_simple

file: `star_formation.feedback.spheroids.fixed.F90`
Description: Implementation of a fixed fraction outflow rate due to star formation feedback in galactic spheroids.
Code lines: 85

function: `fixedconstructorinternal`
Description: Internal constructor for the fixed star formation feedback from spheroids class.
Code lines: 8
Contained by: file star_formation.feedback.spheroids.fixed.F90

function: `fixedconstructorparameters`
Description: Constructor for the fixed fraction star formation feedback in spheroids class which takes a parameter set as input.
Code lines: 20
Contained by: file star_formation.feedback.spheroids.fixed.F90
Modules used: galacticus_error; input_parameters

function: `fixedoutflowrate`
Description: Returns the outflow rate (in $M_\odot$ Gyr^{-1}) for star formation in the galactic spheroid of `node`. Assumes a fixed ratio of outflow rate to star formation rate.
Code lines: 12
Contained by: file star_formation.feedback.spheroids.fixed.F90
Modules used: stellar_feedback

interface: `starformationfeedbackspheroidsfixed`
Description: Constructors for the fixed fraction star formation feedback in spheroids class.

Code lines: 4
Contained by: file star_formation.feedback.spheroids.fixed.F90

file: star_formation.feedback.spheroids.power_law.F90
Description: Implementation of a power-law outflow rate due to star formation feedback in galactic spheroids.
Code lines: 137

function: powerlawconstructorinternal
Description: Internal constructor for the power-law star formation feedback from spheroids class.
Code lines: 14
Contained by: file star_formation.feedback.spheroids.power_law.F90
Modules used: galacticus_error

function: powerlawconstructorparameters
Description: Constructor for the power-law star formation feedback in spheroids class which takes a parameter set as input.
Code lines: 27
Contained by: file star_formation.feedback.spheroids.power_law.F90
Modules used: galacticus_error

function: powerlawoutflowrate
Description: Returns the outflow rate (in $M_\odot$ Gyr^{-1}) for star formation in the galactic spheroid of thisNode. The outflow rate is given by

$$\dot{M}_\mathrm{outflow} = \left(\frac{V_\mathrm{spheroid,outflow}}{V_\mathrm{spheroid}} \right)^{\alpha_\mathrm{spheroid,outflow}}, \tag{18.43}$$

where $V_\mathrm{spheroid,outflow}$(=velocityCharacteristic) is the velocity scale at which outflow rate equals star formation rate and $\alpha_{spheroid,outflow}$(=exponent) controls the scaling with velocity. Note that the velocity V_spheroid is whatever characteristic value returned by the spheroid method. This scaling is functionally similar to that adopted by Cole et al. [2000], but that they specifically used the circular velocity at half-mass radius.
Code lines: 29
Contained by: file star_formation.feedback.spheroids.power_law.F90
Modules used: galacticus_nodes stellar_feedback

function: powerlawvelocitycharacteristic
Description: Return the characteristic velocity for power-law feedback models in spheroids. In this case the characteristic velocity is a constant.
Code lines: 10
Contained by: file star_formation.feedback.spheroids.power_law.F90

interface: starformationfeedbackspheroidspowerlaw
Description: Constructors for the power-law star formation feedback in spheroids class.
Code lines: 4
Contained by: file star_formation.feedback.spheroids.power_law.F90

file: star_formation.feedback.spheroids.power_law.redshift_scaling.F90

Description: Implementation of a power-law outflow rate due to star formation feedback in galactic spheroids in which the characteristic velocity scales as a power of $(1+z)$.
Code lines: 100
Modules used: cosmology_functions

function: powerlawredshiftscalingconstructorinternal
Description: Internal constructor for the power-law redshift-scaling star formation feedback from spheroids class.
Code lines: 10
Contained by: file star_formation.feedback.spheroids.power_law.redshift_scaling.F90

function: powerlawredshiftscalingconstructorparameters
Description: Constructor for the power-law redshift-scaling star formation feedback in spheroids class which takes a parameter set as input.
Code lines: 20
Contained by: file star_formation.feedback.spheroids.power_law.redshift_scaling.F90

subroutine: powerlawredshiftscalingdestructor
Description: Destructor for the power-law redshift-scaling feedback from star formation in spheroids class.
Code lines: 7
Contained by: file star_formation.feedback.spheroids.power_law.redshift_scaling.F90

function: powerlawredshiftscalingvelocitycharacteristic
Description: Return the characteristic velocity for power-law feedback models in spheroids. In this case the characteristic velocity is a constant.
Code lines: 12
Contained by: file star_formation.feedback.spheroids.power_law.redshift_scaling.F90
Modules used: galacticus_nodes

interface: starformationfeedbackspheroidspowerlawredshiftscaling
Description: Constructors for the power-law redshift-scaling star formation feedback in spheroids class.
Code lines: 4
Contained by: file star_formation.feedback.spheroids.power_law.redshift_scaling.F90

file: star_formation.feedback.spheroids.velocity_maximum_scaling.F90
Description: Implementation of an outflow rate due to star formation feedback in galactic spheroids which scales with peak halo velocity.
Code lines: 156
Modules used: cosmology_functions dark_matter_profiles_dmo math_exponentiation

interface: starformationfeedbackspheroidsvlctymxsclng
Description: Constructors for the velocity maximum scaling fraction star formation feedback in spheroids class.
Code lines: 4
Contained by: file star_formation.feedback.spheroids.velocity_maximum_scaling.F90

function: vlctymxsclngconstructorinternal
Description: Internal constructor for the halo scaling star formation feedback from spheroids class.
Code lines: 22

Contained by: file star_formation.feedback.spheroids.velocity_maximum_scaling.F90
Modules used: stellar_feedback

function: vlctymxsclngconstructorparameters
Description: Constructor for the velocity maximum scaling fraction star formation feedback in spheroids class which takes a parameter set as input.
Code lines: 42
Contained by: file star_formation.feedback.spheroids.velocity_maximum_scaling.F90
Modules used: galacticus_error

subroutine: vlctymxsclngdestructor
Description: Destructor for the velocity maximum scaling feedback from star formation in spheroids class.
Code lines: 8
Contained by: file star_formation.feedback.spheroids.velocity_maximum_scaling.F90

function: vlctymxsclngoutflowrate
Description: Returns the outflow rate (in $M_\odot$ Gyr^{-1}) for star formation in the galactic spheroid of node.
Code lines: 29
Contained by: file star_formation.feedback.spheroids.velocity_maximum_scaling.F90
Modules used: galacticus_nodes

file: star_formation.feedback_expulsion.disks.F90
Description: Contains a module which provides a class that implements expulsive feedback from star formation in disks.
Code lines: 41

module: star_formation_feedback_expulsion_disks
Description: Provides a class that implements calculations of expulsive feedback from star formation in disks.
Code lines: 19
Contained by: file star_formation.feedback_expulsion.disks.F90
Modules used: galacticus_nodes
Used by: subroutine galacticus_function_-classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; module node_component_disk_standard

file: star_formation.feedback_expulsion.disks.superwind.F90
Description: Implementation of a "superwind" expulsive outflow rate due to star formation feedback in galactic disks.
Code lines: 110

interface: starformationexpulsivefeedbackdiskssuperwind
Description: Constructors for the superwind expulsive star formation feedback in disks class.
Code lines: 4
Contained by: file star_formation.feedback_expulsion.disks.superwind.F90

function: superwindconstructorinternal
Description: Internal constructor for the superwind expulsive star formation feedback from disks class.
Code lines: 8

Contained by: file star_formation.feedback_expulsion.disks.superwind.F90

function: superwindconstructorparameters
Description: Constructor for the superwind expulsive star formation feedback in disks class which takes a parameter set as input.
Code lines: 27
Contained by: file star_formation.feedback_expulsion.disks.superwind.F90
Modules used: input_parameters

function: superwindoutflowrate
Description: Returns the expulsive outflow rate (in $M_\odot$ Gyr^{-1}) for star formation in the galactic disk of node. The outflow rate is given by

$$\dot{M}_{\rm outflow} = f_{\rm SW,0} \begin{cases} 1 & \text{if } V_{\rm disk} < V_{\rm disk,SW} \\ (V_{\rm disk,SW}/V_{\rm disk})^2 & \text{if } V_{\rm disk} \geq V_{\rm disk,SW} \end{cases} , \tag{18.44}$$

where $V_{\rm disk,SW}$ =[velocityCharacteristic] and $f_{\rm SW,0}$ =[massLoadnig]. Note that the velocity $V_{\rm disk}$ is whatever characteristic value returned by the disk method. This scaling is functionally similar to that adopted by Cole et al. [2000] and Baugh et al. [2005], except that they specifically used the circular velocity at half-mass radius.
Code lines: 30
Contained by: file star_formation.feedback_expulsion.disks.superwind.F90
Modules used: galacticus_nodes stellar_feedback

file: star_formation.feedback_expulsion.disks.zero.F90
Description: Implementation of a zero expulsive outflow rate due to star formation feedback in galactic disks.
Code lines: 61

interface: starformationexpulsivefeedbackdiskszero
Description: Constructors for the superwind expulsive star formation feedback in disks class.
Code lines: 3
Contained by: file star_formation.feedback_expulsion.disks.zero.F90

function: zeroconstructorparameters
Description: Constructor for the superwind expulsive star formation feedback in disks class which takes a parameter set as input.
Code lines: 10
Contained by: file star_formation.feedback_expulsion.disks.zero.F90
Modules used: input_parameters

function: zerooutflowrate
Description: Returns a zero expulsive outflow rate from disks
Code lines: 10
Contained by: file star_formation.feedback_expulsion.disks.zero.F90

file: star_formation.feedback_expulsion.spheroids.F90
Description: Contains a module which provides a class that implements expulsive feedback from star formation in spheroids.
Code lines: 41

module: `star_formation_feedback_expulsion_spheroids`

Description: Provides a class that implements calculations of expulsive feedback from star formation in spheroids.
Code lines: 19
Contained by: file `star_formation.feedback_expulsion.spheroids.F90`
Modules used: `galacticus_nodes`
Used by: subroutine `galacticus_function_classes_destroy`; subroutine `galacticus_state_retrieve`; subroutine `galacticus_state_store`; module `node_component_spheroid_standard`

file: `star_formation.feedback_expulsion.spheroids.superwind.F90`

Description: Implementation of a "superwind" expulsive outflow rate due to star formation feedback in galactic spheroids.
Code lines: 110

interface: `starformationexpulsivefeedbackspheroidssuperwind`

Description: Constructors for the superwind expulsive star formation feedback in spheroids class.
Code lines: 4
Contained by: file `star_formation.feedback_expulsion.spheroids.superwind.F90`

function: `superwindconstructorinternal`

Description: Internal constructor for the superwind expulsive star formation feedback from spheroids class.
Code lines: 8
Contained by: file `star_formation.feedback_expulsion.spheroids.superwind.F90`

function: `superwindconstructorparameters`

Description: Constructor for the superwind expulsive star formation feedback in spheroids class which takes a parameter set as input.
Code lines: 27
Contained by: file `star_formation.feedback_expulsion.spheroids.superwind.F90`
Modules used: `input_parameters`

function: `superwindoutflowrate`

Description: Returns the expulsive outflow rate (in $M_\odot$ Gyr^{-1}) for star formation in the galactic spheroid of `node`. The outflow rate is given by

$$\dot{M}_\mathrm{outflow} = f_\mathrm{SW,0} \left\{ \begin{array}{ll} 1 & \hbox{if } V_\mathrm{spheroid} < V_\mathrm{spheroid,SW} \\ (V_\mathrm{spheroid,SW}/V_\mathrm{spheroid})^2 & \hbox{if } V_\mathrm{spheroid} \geq V_\mathrm{spheroid,SW} \end{array} \right. , \tag{18.45}$$

where $V_\mathrm{spheroid,SW}$ =`[velocityCharacteristic]` and $f_\mathrm{SW,0}$ =`[massLoadnig]`. Note that the velocity V_spheroid is whatever characteristic value returned by the spheroid method. This scaling is functionally similar to that adopted by Cole et al. [2000] and Baugh et al. [2005], except that they specifically used the circular velocity at half-mass radius.
Code lines: 30
Contained by: file `star_formation.feedback_expulsion.spheroids.superwind.F90`
Modules used: `galacticus_nodes`; `stellar_feedback`

file: `star_formation.feedback_expulsion.spheroids.zero.F90`

Description: Implementation of a zero expulsive outflow rate due to star formation feedback in galactic spheroids.
Code lines: 61

interface: `starformationexpulsivefeedbackspheroidszero`
Description: Constructors for the superwind expulsive star formation feedback in spheroids class.
Code lines: 3
Contained by: file star_formation.feedback_expulsion.spheroids.zero.F90

function: `zeroconstructorparameters`
Description: Constructor for the superwind expulsive star formation feedback in spheroids class which takes a parameter set as input.
Code lines: 10
Contained by: file star_formation.feedback_expulsion.spheroids.zero.F90
Modules used: input_parameters

function: `zerooutflowrate`
Description: Returns a zero expulsive outflow rate from spheroids
Code lines: 10
Contained by: file star_formation.feedback_expulsion.spheroids.zero.F90

file: `star_formation.histories.F90`
Description: Contains a module which implements a class for computation and output of star formation histories for galaxies.
Code lines: 75

module: `star_formation_histories`
Description: Implements a class for computation and output of star formation histories for galaxies.
Code lines: 53
Contained by: file star_formation.histories.F90
Modules used: abundances_structure, galacticus_nodes, histories, iso_c_binding, kind_numbers
Used by: subroutine galacticus_function_classes_destroy, subroutine galacticus_state_retrieve, subroutine galacticus_state_store, module node_component_disk_standard, module node_component_spheroid_standard

file: `star_formation.histories.in_situ.F90`
Description: Contains a module which implements a star formation histories class which records *in situ* star formation.
Code lines: 428
Modules used: output_times

subroutine: `insituautohook`
Description: Attach to the satellite merging event hook.
Code lines: 8
Contained by: file star_formation.histories.in_situ.F90
Modules used: events_hooks

function: `insituconstructorinternal`
Description: Internal constructor for the "inSitu" star formation history class.
Code lines: 9
Contained by: file `star_formation.histories.in_situ.F90`

function: `insituconstructorparameters`
Description: Constructor for the "inSitu" star formation history class which takes a parameter set as input.
Code lines: 38
Contained by: file `star_formation.histories.in_situ.F90`
Modules used: `input_parameters`

subroutine: `insitucreate`
Description: Create the history required for storing star formation history.
Code lines: 17
Contained by: file `star_formation.histories.in_situ.F90`
Modules used: `galacticus_nodes`

subroutine: `insitudestructor`
Description: Destructor for the `inSitu` star formation histories class.
Code lines: 7
Contained by: file `star_formation.histories.in_situ.F90`

subroutine: `insitumake`
Description: Create the history required for storing star formation history.
Code lines: 133
Contained by: file `star_formation.histories.in_situ.F90`
Modules used: `galacticus_error` `numerical_ranges`

subroutine: `insituoutput`
Description: Output the star formation history for `node`.
Code lines: 59
Contained by: file `star_formation.histories.in_situ.F90`
Modules used: `galacticus_hdf5` `galacticus_nodes` `io_hdf5` `string_handling`

subroutine: `insiturate`
Description: Set the rate the star formation history for `node`.
Code lines: 22
Contained by: file `star_formation.histories.in_situ.F90`
Modules used: `arrays_search` `galacticus_nodes`

subroutine: `insitusatellitemerger`
Description: Zero any in-situ star formation history for galaxy about to merge.
Code lines: 29
Contained by: file `star_formation.histories.in_situ.F90`
Modules used: `galacticus_error` `galacticus_nodes`

subroutine: `insituscales`
Description: Set the scalings for error control on the absolute values of star formation histories.
Code lines: 20
Contained by: file `star_formation.histories.in_situ.F90`
Modules used: `memory_management`

type: `insitutimesteprange`
Code lines: 5
Contained by: file `star_formation.histories.in_situ.F90`

interface: `starformationhistoryinsitu`
Description: Constructors for the "inSitu" star formation history class.
Code lines: 4
Contained by: file `star_formation.histories.in_situ.F90`

file: `star_formation.histories.metallicity_split.F90`
Description: Contains a module which implements a star formation histories class which records star formation split by metallicity.
Code lines: 470
Modules used: `output_times`

function: `metallicitysplitconstructorinternal`
Description: Internal constructor for the "metallicitySplit" star formation history class.
Code lines: 34
Contained by: file `star_formation.histories.metallicity_split.F90`
Modules used: `galacticus_error` `numerical_ranges`

function: `metallicitysplitconstructorparameters`
Description: Constructor for the "metallicitySplit" star formation history class which takes a parameter set as input.
Code lines: 84
Contained by: file `star_formation.histories.metallicity_split.F90`
Modules used: `input_parameters`

subroutine: `metallicitysplitcreate`
Description: Create the history required for storing star formation history.
Code lines: 16
Contained by: file `star_formation.histories.metallicity_split.F90`
Modules used: `galacticus_nodes`

subroutine: `metallicitysplitdestructor`
Description: Destructor for the `metallicitySplit` star formation histories class.
Code lines: 7
Contained by: file `star_formation.histories.metallicity_split.F90`

subroutine: `metallicitysplitmake`
Description: Create the history required for storing star formation history.
Code lines: 132

Contained by: file star_formation.histories.metallicity_split.F90
Modules used: galacticus_error numerical_ranges

subroutine: metallicitysplitoutput
Description: Output the star formation history for node.
Code lines: 61
Contained by: file star_formation.histories.metallicity_split.F90
Modules used: galacticus_hdf5 galacticus_nodes io_hdf5 string_handling

subroutine: metallicitysplitrate
Description: Set the rate the star formation history for node.
Code lines: 31
Contained by: file star_formation.histories.metallicity_split.F90
Modules used: abundances_structure arrays_search galacticus_nodes

subroutine: metallicitysplitscales
Description: Set the scalings for error control on the absolute values of star formation histories.
Code lines: 19
Contained by: file star_formation.histories.metallicity_split.F90

type: metallicitysplittimesteprange
Code lines: 5
Contained by: file star_formation.histories.metallicity_split.F90

interface: starformationhistorymetallicitysplit
Description: Constructors for the "metallicitySplit" star formation history class.
Code lines: 4
Contained by: file star_formation.histories.metallicity_split.F90

file: star_formation.histories.null.F90
Description: Contains a module which implements a null star formation histories class.
Code lines: 109

function: nullconstructorparameters
Description: Constructor for the "null" star formation history class which takes a parameter set as input.
Code lines: 10
Contained by: file star_formation.histories.null.F90
Modules used: input_parameters

subroutine: nullcreate
Description: Create the history required for storing star formation history.
Code lines: 11
Contained by: file star_formation.histories.null.F90

subroutine: nulloutput
Description: Output the star formation history for node.
Code lines: 14

Contained by: file star_formation.histories.null.F90

subroutine: nullrate
Description: Set the rate the star formation history for node.
Code lines: 13
Contained by: file star_formation.histories.null.F90

subroutine: nullscales
Description: Set the scalings for error control on the absolute values of star formation histories.
Code lines: 11
Contained by: file star_formation.histories.null.F90

interface: starformationhistorynull
Description: Constructors for the "null" star formation history class.
Code lines: 3
Contained by: file star_formation.histories.null.F90

file: star_formation.rate_surface_density.disks.Blitz2006.F90
Description: Implementation of the Blitz and Rosolowsky [2006] star formation rate surface density law for galactic disks.
Code lines: 246
Modules used: kind_numbers

subroutine: blitz2006autohook
Description: Attach to the calculation reset event.
Code lines: 8
Contained by: file star_formation.rate_surface_density.disks.Blitz2006.F90
Modules used: events_hooks

subroutine: blitz2006calculationreset
Description: Reset the Kennicutt-Schmidt relation calculation.
Code lines: 9
Contained by: file star_formation.rate_surface_density.disks.Blitz2006.F90

function: blitz2006constructorinternal
Description: Internal constructor for the blitz2006 star formation surface density rate from disks class.
Code lines: 20
Contained by: file star_formation.rate_surface_density.disks.Blitz2006.F90
Modules used: galacticus_error numerical_constants_astronomical numerical_constants_physical numerical_constants_prefixes

function: blitz2006constructorparameters
Description: Constructor for the blitz2006 star formation surface density rate in disks class which takes a parameter set as input.
Code lines: 80
Contained by: file star_formation.rate_surface_density.disks.Blitz2006.F90

subroutine: blitz2006destructor
Description: Destructor for the blitz2006 cooling radius class.
Code lines: 8

Contained by: file star_formation.rate_surface_density.disks.Blitz2006.F90
Modules used: events_hooks

function: blitz2006rate
Description: Returns the star formation rate surface density (in $M_\odot$ Gyr^{-1} Mpc^{-2}) for star formation in the galactic disk of node. The disk is assumed to obey the Blitz and Rosolowsky [2006] star formation rule.
Code lines: 54
Contained by: file star_formation.rate_surface_density.disks.Blitz2006.F90
Modules used: abundances_structure, galactic_structure_options, galactic_structure_surface_densities, galacticus_nodes, numerical_constants_math, numerical_constants_physical

interface: starformationratesurfacedensitydisksblitz2006
Description: Constructors for the blitz2006 star formation surface density rate in disks class.
Code lines: 4
Contained by: file star_formation.rate_surface_density.disks.Blitz2006.F90

file: star_formation.rate_surface_density.disks.F90
Description: Contains a module which provides a class that implements surface density rates of star formation in disks.
Code lines: 65

module: star_formation_rate_surface_density_disks
Description: Provides a class that implements surface density rates of star formation in disks.
Code lines: 43
Contained by: file star_formation.rate_surface_density.disks.F90
Modules used: galacticus_nodes
Used by: subroutine galacticus_function_classes_destroy, subroutine galacticus_state_retrieve, subroutine galacticus_state_store, file star_formation.feedback.disks.Creasey2012.F90, file star_formation.timescales.disks.integrated_surface_density.F90

file: star_formation.rate_surface_density.disks.Kennicutt-Schmidt.F90
Description: Implementation of a the Kennicutt-Schmidt star formation rate surface density for galactic disks.
Code lines: 240
Modules used: kind_numbers

subroutine: kennicuttschmidtautohook
Description: Attach to the calculation reset event.
Code lines: 8
Contained by: file star_formation.rate_surface_density.disks.Kennicutt-Schmidt.F90
Modules used: events_hooks

subroutine: kennicuttschmidtcalculationreset
Description: Reset the Kennicutt-Schmidt relation calculation.

Code lines: 9
Contained by: file star_formation.rate_surface_density.disks.Kennicutt-Schmidt.F90

function: kennicuttschmidtconstructorinternal
Description: Internal constructor for the kennicuttSchmidt star formation surface density rate from disks class.
Code lines: 14
Contained by: file star_formation.rate_surface_density.disks.Kennicutt-Schmidt.F90
Modules used: numerical_constants_prefixes

function: kennicuttschmidtconstructorparameters
Description: Constructor for the kennicuttSchmidt star formation surface density rate in disks class which takes a parameter set as input.
Code lines: 70
Contained by: file star_formation.rate_surface_density.disks.Kennicutt-Schmidt.F90
Modules used: galacticus_error

subroutine: kennicuttschmidtdestructor
Description: Destructor for the kennicuttSchmidt cooling radius class.
Code lines: 8
Contained by: file star_formation.rate_surface_density.disks.Kennicutt-Schmidt.F90
Modules used: events_hooks

function: kennicuttschmidtrate
Description: Returns the star formation rate surface density (in $M_\odot$ Gyr^{-1} Mpc^{-2}) for star formation in the galactic disk of node. The disk is assumed to obey the Kennicutt-Schmidt law:

$$\Sigma_\star = A \left(x_\mathrm{H} \frac{\Sigma_\mathrm{gas}}{M_\odot \mathrm{pc}^{-2}} \right)^N, \tag{18.46}$$

where $A =$[normalization] and $N =$[exponent]. Optionally, star formation is truncated for gas surface densities below a critical density of:

$$\Sigma_\mathrm{crit} = \frac{q_\mathrm{crit} \kappa \sigma_\mathrm{gas}}{\pi \mathrm{G}}, \tag{18.47}$$

where κ is the epicyclic frequency in the disk, σ_gas is the velocity dispersion of gas in the disk and $q_\mathrm{crit} =$[toomreParameterCritical] is a dimensionless constant of order unity which controls where the critical density occurs. σ_gas is assumed to be a constant equal to [velocityDispersionDiskGas] and the disk is assumed to have a flat rotation curve such that $\kappa = \sqrt{2} V / R$.
Code lines: 62
Contained by: file star_formation.rate_surface_density.disks.Kennicutt-Schmidt.F90
Modules used: abundances_structure, galactic_structure_options, galactic_structure_surface_densities, galacticus_nodes, numerical_constants_math, numerical_constants_physical

interface: starformationratesurfacedensitydiskskennicuttschmidt
Description: Constructors for the kennicuttSchmidt star formation surface density rate in disks class.
Code lines: 4
Contained by: file star_formation.rate_surface_density.disks.Kennicutt-Schmidt.F90

file: `star_formation.rate_surface_density.disks.Krumholz2009.F90`
Description: Implementation of the Krumholz et al. [2009] star formation rate surface density law for galactic disks.
Code lines: 439
Modules used: `abundances_structure` `kind_numbers` `math_exponentiation` `tables`

subroutine: `krumholz2009autohook`
Description: Attach to the calculation reset event.
Code lines: 8
Contained by: file `star_formation.rate_surface_density.disks.Krumholz2009.F90`
Modules used: `events_hooks`

subroutine: `krumholz2009calculationreset`
Description: Reset the Kennicutt-Schmidt relation calculation.
Code lines: 9
Contained by: file `star_formation.rate_surface_density.disks.Krumholz2009.F90`

subroutine: `krumholz2009computefactors`
Description: Compute constant factors needed in the Krumholz et al. [2009] star formation rule.
Code lines: 34
Contained by: file `star_formation.rate_surface_density.disks.Krumholz2009.F90`
Modules used: `galacticus_nodes` `numerical_constants_prefixes`

function: `krumholz2009constructorinternal`
Description: Internal constructor for the `krumholz2009` star formation surface density rate from disks class.
Code lines: 32
Contained by: file `star_formation.rate_surface_density.disks.Krumholz2009.F90`
Modules used: `table_labels`

function: `krumholz2009constructorparameters`
Description: Constructor for the `krumholz2009` star formation surface density rate in disks class which takes a parameter set as input.
Code lines: 49
Contained by: file `star_formation.rate_surface_density.disks.Krumholz2009.F90`
Modules used: `galacticus_error`

function: `krumholz2009criticaldensityroot`
Description: Root function used in finding the radius in a disk where the surface density equals the critical surface density in the Krumholz et al. [2009] star formation rate model.
Code lines: 10
Contained by: file `star_formation.rate_surface_density.disks.Krumholz2009.F90`

subroutine: `krumholz2009destructor`
Description: Destructor for the `krumholz2009` star formation surface density rate from disks class.
Code lines: 9
Contained by: file `star_formation.rate_surface_density.disks.Krumholz2009.F90`
Modules used: `events_hooks`

function: `krumholz2009intervals`
Description: Returns intervals to use for integrating the Krumholz et al. [2009] star formation rate over a galactic disk.
Code lines: 50
Contained by: file `star_formation.rate_surface_density.disks.Krumholz2009.F90`
Modules used: `root_finder`

function: `krumholz2009molecularfractionfast`
Description: Fast (but less accurate at low molecular fraction) fitting function from McKee and Krumholz [2010] for the molecular hydrogen fraction.
Code lines: 13
Contained by: file `star_formation.rate_surface_density.disks.Krumholz2009.F90`

function: `krumholz2009molecularfractionslow`
Description: Slow (but more accurate at low molecular fraction) fitting function from Krumholz et al. [2009] for the molecular hydrogen fraction.
Code lines: 20
Contained by: file `star_formation.rate_surface_density.disks.Krumholz2009.F90`

function: `krumholz2009rate`
Description: Returns the star formation rate surface density (in $M_\odot$ Gyr^{-1} Mpc^{-2}) for star formation in the galactic disk of `node`. The disk is assumed to obey the Krumholz et al. [2009] star formation rule.
Code lines: 42
Contained by: file `star_formation.rate_surface_density.disks.Krumholz2009.F90`

subroutine: `krumholz2009surfacedensityfactors`
Description: Compute surface density and related quantities needed for the Krumholz et al. [2009] star formation rate model.
Code lines: 16
Contained by: file `star_formation.rate_surface_density.disks.Krumholz2009.F90`
Modules used: `galactic_structure_options` `galactic_structure_surface_densities`

function: `krumholz2009unchanged`
Description: Determine if the surface rate density of star formation is unchanged.
Code lines: 33
Contained by: file `star_formation.rate_surface_density.disks.Krumholz2009.F90`
Modules used: `galacticus_nodes`

interface: `starformationratesurfacedensitydiskskrumholz2009`
Description: Constructors for the `krumholz2009` star formation surface density rate in disks class.
Code lines: 4
Contained by: file `star_formation.rate_surface_density.disks.Krumholz2009.F90`

file: `star_formation.rate_surface_density.disks.extended_Schmidt.F90`
Description: Implementation of the extended Schmidt star formation rate surface density law of Shi et al. [2011] for galactic disks.
Code lines: 193

subroutine: `extendedschmidtautohook`

Description: Attach to the calculation reset event.
Code lines: 8
Contained by: file star_formation.rate_surface_density.disks.extended_Schmidt.F90
Modules used: events_hooks

subroutine: extendedschmidtcalculationreset
Description: Reset the Kennicutt-Schmidt relation calculation.
Code lines: 9
Contained by: file star_formation.rate_surface_density.disks.extended_Schmidt.F90

function: extendedschmidtconstructorinternal
Description: Internal constructor for the extendedSchmidt star formation surface density rate from disks class.
Code lines: 13
Contained by: file star_formation.rate_surface_density.disks.extended_Schmidt.F90
Modules used: numerical_constants_prefixes

function: extendedschmidtconstructorparameters
Description: Constructor for the extendedSchmidt star formation surface density rate in disks class which takes a parameter set as input.
Code lines: 41
Contained by: file star_formation.rate_surface_density.disks.extended_Schmidt.F90
Modules used: galacticus_error

subroutine: extendedschmidtdestructor
Description: Destructor for the extendedSchmidt cooling radius class.
Code lines: 8
Contained by: file star_formation.rate_surface_density.disks.extended_Schmidt.F90
Modules used: events_hooks

function: extendedschmidtrate
Description: Returns the star formation rate surface density (in $M_\odot$ Gyr^{-1} Mpc^{-2}) for star formation in the galactic disk of node. The disk is assumed to obey the extended Schmidt law of Shi et al. [2011]:

$$\dot{\Sigma}_\star = A \left(x_\mathrm{H} \frac{\Sigma_\mathrm{gas}}{M_\odot \mathrm{pc}^{-2}} \right)^{N_1} \left(\frac{\Sigma_\star}{M_\odot \mathrm{pc}^{-2}} \right)^{N_2}, \tag{18.48}$$

where $A =$[normalization] and $N_1 =$[exponentGas]. $N_2 =$[exponentStars].
Code lines: 47
Contained by: file star_formation.rate_surface_density.disks.extended_Schmidt.F90
Modules used: abundances_structure, galactic_structure_options, galactic_structure_surface_densities, galacticus_nodes

interface: starformationratesurfacedensitydisksextendedschmidt
Description: Constructors for the extendedSchmidt star formation surface density rate in disks class.
Code lines: 4
Contained by: file star_formation.rate_surface_density.disks.extended_Schmidt.F90

file: star_formation.timescales.disks.Baugh2005.F90
Description: Implementation of the Baugh et al. [2005] timescale for star formation in galactic disks

Code lines: 133
Modules used: cosmology_functions

function: baugh2005constructorinternal
Description: Internal constructor for the baugh2005 timescale for star formation in disks class.
Code lines: 10
Contained by: file star_formation.timescales.disks.Baugh2005.F90

function: baugh2005constructorparameters
Description: Constructor for the baugh2005 timescale for star formation in disks class which takes a parameter set as input.
Code lines: 42
Contained by: file star_formation.timescales.disks.Baugh2005.F90
Modules used: input_parameters

subroutine: baugh2005destructor
Description: Destructor for the baugh2005 timescale for star formation in disks class.
Code lines: 7
Contained by: file star_formation.timescales.disks.Baugh2005.F90

function: baugh2005timescale
Description: Returns the timescale (in Gyr) for star formation in the galactic disk of node in the halo scaling timescale model.
Code lines: 21
Contained by: file star_formation.timescales.disks.Baugh2005.F90
Modules used: galacticus_nodes

interface: starformationtimescaledisksbaugh2005
Description: Constructors for the baugh2005 timescale for star formation in disks class.
Code lines: 4
Contained by: file star_formation.timescales.disks.Baugh2005.F90

file: star_formation.timescales.disks.F90
Description: Contains a module which provides a class that implements timescales for star formation in disks.
Code lines: 42

module: star_formation_timescales_disks
Description: Provides a class that implements calculations of timescales for star formation in disks.
Code lines: 20
Contained by: file star_formation.timescales.disks.F90
Modules used: galacticus_nodes
Used by: subroutine galacticus_function_-classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; module node_component_disk_standard; module node_component_disk_very_simple

file: star_formation.timescales.disks.dynamical_time.F90
Description: Implementation of a timescale for star formation in galactic disks which scales with the disk dynamical time.

Code lines: 131

function: `dynamicaltimeconstructorinternal`
Description: Internal constructor for the `dynamicalTime` timescale for star formation in disks class.
Code lines: 12
Contained by: file `star_formation.timescales.disks.dynamical_time.F90`
Modules used: `array_utilities` `galacticus_error` `galacticus_nodes`

function: `dynamicaltimeconstructorparameters`
Description: Constructor for the `dynamicalTime` timescale for star formation in disks class which takes a parameter set as input.
Code lines: 39
Contained by: file `star_formation.timescales.disks.dynamical_time.F90`
Modules used: `input_parameters`

function: `dynamicaltimetimescale`
Description: Returns the timescale (in Gyr) for star formation in the galactic disk of `node`. The timescale is given by

$$\tau_\star = \epsilon_\star^{-1} \tau_{\rm dynamical,disk} \left(\frac{V_{\rm disk}}{200{\rm km/s}} \right)^{\alpha_\star}, \tag{18.49}$$

where $\epsilon_\star$(=`effiency`) is a star formation efficiency and $\alpha_\star$(=`exponentVelocity`) controls the scaling with velocity. Note that $\tau_{\rm dynamical,disk} = R_{\rm disk}/V_{\rm disk}$ where the radius and velocity are whatever characteristic values returned by the disk method. This scaling is functionally similar to that adopted by Cole et al. [2000], but that they specifically used the half-mass radius and circular velocity at that radius.
Code lines: 35
Contained by: file `star_formation.timescales.disks.dynamical_time.F90`
Modules used: `galacticus_nodes` `numerical_constants_astronomical`

interface: `starformationtimescaledisksdynamicaltime`
Description: Constructors for the `dynamicalTime` timescale for star formation in disks class.
Code lines: 4
Contained by: file `star_formation.timescales.disks.dynamical_time.F90`

file: `star_formation.timescales.disks.fixed.F90`
Description: Implementation of a fixed timescale for star formation in galactic disks.
Code lines: 84

function: `fixedconstructorinternal`
Description: Internal constructor for the `fixed` timescale for star formation in disks class.
Code lines: 8
Contained by: file `star_formation.timescales.disks.fixed.F90`

function: `fixedconstructorparameters`
Description: Constructor for the `fixed` timescale for star formation in disks class which takes a parameter set as input.
Code lines: 21
Contained by: file `star_formation.timescales.disks.fixed.F90`
Modules used: `input_parameters`

function: `fixedtimescale`
Description: Returns the timescale (in Gyr) for star formation in the galactic disk of `node`, assuming a fixed timecale.
Code lines: 10
Contained by: file star_formation.timescales.disks.fixed.F90

interface: `starformationtimescaledisksfixed`
Description: Constructors for the `fixed` timescale for star formation in disks class.
Code lines: 4
Contained by: file star_formation.timescales.disks.fixed.F90

file: `star_formation.timescales.disks.halo_scaling.F90`
Description: Implementation of a timescale for star formation feedback in galactic disks which scales with the circular velocity of the host halo.
Code lines: 198
Modules used: cosmology_functions dark_matter_halo_scales

subroutine: `haloscalingautohook`
Description: Attach to the calculation reset event.
Code lines: 8
Contained by: file star_formation.timescales.disks.halo_scaling.F90
Modules used: events_hooks

subroutine: `haloscalingcalculationreset`
Description: Reset the halo scaling disk star formation timescale calculation.
Code lines: 9
Contained by: file star_formation.timescales.disks.halo_scaling.F90

function: `haloscalingconstructorinternal`
Description: Internal constructor for the `haloScaling` timescale for star formation in disks class.
Code lines: 18
Contained by: file star_formation.timescales.disks.halo_scaling.F90

function: `haloscalingconstructorparameters`
Description: Constructor for the `haloScaling` timescale for star formation feedback in disks class which takes a parameter set as input.
Code lines: 45
Contained by: file star_formation.timescales.disks.halo_scaling.F90
Modules used: input_parameters

subroutine: `haloscalingdestructor`
Description: Destructor for the `haloScaling` timescale for star formation in disks class.
Code lines: 10
Contained by: file star_formation.timescales.disks.halo_scaling.F90
Modules used: events_hooks

function: `haloscalingtimescale`
Description: Returns the timescale (in Gyr) for star formation in the galactic disk of `node` in the halo scaling timescale model.
Code lines: 36

Contained by: file star_formation.timescales.disks.halo_scaling.F90
Modules used: galacticus_nodes

interface: starformationtimescalediskshaloscaling
Description: Constructors for the haloScaling timescale for star formation in disks class.
Code lines: 4
Contained by: file star_formation.timescales.disks.halo_scaling.F90

file: star_formation.timescales.disks.integrated_surface_density.F90
Description: Implementation of a timescale for star formation in galactic disks which computes the timescale by integrating a star formation rate over the disk.
Code lines: 164
Modules used: star_formation_rate_surface_density_-disks

function: intgrtdsurfacedensityconstructorinternal
Description: Internal constructor for the intgrtdSurfaceDensity timescale for star formation in disks class.
Code lines: 9
Contained by: file star_formation.timescales.disks.integrated_surface_density.F90

function: intgrtdsurfacedensityconstructorparameters
Description: Constructor for the intgrtdSurfaceDensity timescale for star formation in disks class which takes a parameter set as input.
Code lines: 23
Contained by: file star_formation.timescales.disks.integrated_surface_density.F90
Modules used: input_parameters

subroutine: intgrtdsurfacedensitydestructor
Description: Destructor for the intgrtdSurfaceDensity timescale for star formation in disks class.
Code lines: 7
Contained by: file star_formation.timescales.disks.integrated_surface_density.F90

function: intgrtdsurfacedensityintegrand
Description: Integrand function for the "integrated surface density" star formation rate calculation.
Code lines: 7
Contained by: file star_formation.timescales.disks.integrated_surface_density.F90

function: intgrtdsurfacedensitytimescale
Description: Returns the timescale (in Gyr) for star formation in the galactic disk of node, by integrating over the surface density of star formation rate.
Code lines: 58
Contained by: file star_formation.timescales.disks.integrated_surface_density.F90
Modules used: fgsl galacticus_nodes numerical_constants_math numerical_integration

interface: starformationtimescalediskssintgrtdsurfacedensity
Description: Constructors for the intgrtdSurfaceDensity timescale for star formation in disks class.
Code lines: 4

Contained by: file star_formation.timescales.disks.integrated_surface_density.F90

file: star_formation.timescales.disks.velocity_maximum_scaling.F90
Description: Implementation of a timescale for star formation in galactic disks which scales with the circular velocity of the host halo.
Code lines: 205
Modules used: cosmology_functions, dark_matter_profiles_dmo, kind_numbers, math_exponentiation

interface: starformationtimescaledisksvelocitymaxscaling
Description: Constructors for the velocityMaxScaling timescale for star formation in disks class.
Code lines: 4
Contained by: file star_formation.timescales.disks.velocity_maximum_scaling.F90

subroutine: velocitymaxscalingautohook
Description: Attach to the calculation reset event.
Code lines: 8
Contained by: file star_formation.timescales.disks.velocity_maximum_scaling.F90
Modules used: events_hooks

subroutine: velocitymaxscalingcalculationreset
Description: Reset the halo scaling disk star formation timescale calculation.
Code lines: 9
Contained by: file star_formation.timescales.disks.velocity_maximum_scaling.F90

function: velocitymaxscalingconstructorinternal
Description: Internal constructor for the velocityMaxScaling timescale for star formation in disks class.
Code lines: 21
Contained by: file star_formation.timescales.disks.velocity_maximum_scaling.F90

function: velocitymaxscalingconstructorparameters
Description: Constructor for the velocityMaxScaling timescale for star formation in disks class which takes a parameter set as input.
Code lines: 46
Contained by: file star_formation.timescales.disks.velocity_maximum_scaling.F90
Modules used: input_parameters

subroutine: velocitymaxscalingdestructor
Description: Destructor for the velocityMaxScaling timescale for star formation in disks class.
Code lines: 10
Contained by: file star_formation.timescales.disks.velocity_maximum_scaling.F90
Modules used: events_hooks

function: velocitymaxscalingtimescale
Description: Returns the timescale (in Gyr) for star formation in the galactic disk of node in the halo scaling timescale model.
Code lines: 36
Contained by: file star_formation.timescales.disks.velocity_maximum_scaling.F90
Modules used: galacticus_nodes

file: `star_formation.timescales.spheroids.F90`
Description: Contains a module which provides a class that implements timescales for star formation in spheroids.
Code lines: 41

module: `star_formation_timescales_spheroids`
Description: Provides a class that implements calculations of timescales for star formation in spheroids.
Code lines: 19
Contained by: file star_formation.timescales.spheroids.F90
Modules used: galacticus_nodes
Used by: subroutine galacticus_function_classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; module node_component_spheroid_standard; module node_component_spheroid_very_simple

file: `star_formation.timescales.spheroids.dynamical_time.F90`
Description: Implementation of a timescale for star formation in galactic spheroids which scales with the spheroid dynamical time.
Code lines: 132

function: `dynamicaltimeconstructorinternal`
Description: Internal constructor for the `dynamicalTime` timescale for star formation in spheroids class.
Code lines: 12
Contained by: file star_formation.timescales.spheroids.dynamical_time.F90
Modules used: array_utilities; galacticus_error; galacticus_nodes

function: `dynamicaltimeconstructorparameters`
Description: Constructor for the `dynamicalTime` timescale for star formation in spheroids class which takes a parameter set as input.
Code lines: 39
Contained by: file star_formation.timescales.spheroids.dynamical_time.F90
Modules used: input_parameters

function: `dynamicaltimetimescale`

Description: Returns the timescale (in Gyr) for star formation in the galactic spheroid of `node`. The timescale is given by

$$\tau_\star = \epsilon_\star^{-1} \tau_{\rm dynamical,spheroid} \left(\frac{V_{\rm spheroid}}{200{\rm km/s}} \right)^{\alpha_\star}, \tag{18.50}$$

where $\epsilon_\star$(=`efficiency`) is a star formation efficiency and $\alpha_\star$(=`exponentVelocity`) controls the scaling with velocity. Note that $\tau_{\rm dynamical,spheroid} = R_{\rm spheroid}/V_{\rm spheroid}$ where the radius and velocity are whatever characteristic values returned by the spheroid method. This scaling is functionally similar to that adopted by Cole et al. [2000], but that they specifically used the half-mass radius and circular velocity at that radius.

Code lines: 36

Contained by: file `star_formation.timescales.spheroids.dynamical_time.F90`

Modules used: `galacticus_nodes` `numerical_constants_astronomical`

interface: `starformationtimescalespheroidsdynamicaltime`

Description: Constructors for the `dynamicalTime` timescale for star formation in spheroids class.
Code lines: 4
Contained by: file star_formation.timescales.spheroids.dynamical_time.F90

file: `star_formation.timescales.spheroids.velocity_maximum_scaling.F90`
Description: Implementation of a timescale for star formation in galactic spheroids which scales with the circular velocity of the host halo.
Code lines: 205
Modules used: cosmology_functions, dark_matter_profiles_dmo, kind_numbers, math_exponentiation

interface: `starformationtimescalespheroidsvelocitymaxscaling`
Description: Constructors for the `velocityMaxScaling` timescale for star formation in spheroids class.
Code lines: 4
Contained by: file star_formation.timescales.spheroids.velocity_maximum_scaling.F90

subroutine: `velocitymaxscalingautohook`
Description: Attach to the calculation reset event.
Code lines: 8
Contained by: file star_formation.timescales.spheroids.velocity_maximum_scaling.F90
Modules used: events_hooks

subroutine: `velocitymaxscalingcalculationreset`
Description: Reset the halo scaling spheroid star formation timescale calculation.
Code lines: 9
Contained by: file star_formation.timescales.spheroids.velocity_maximum_scaling.F90

function: `velocitymaxscalingconstructorinternal`
Description: Internal constructor for the `velocityMaxScaling` timescale for star formation in spheroids class.
Code lines: 21
Contained by: file star_formation.timescales.spheroids.velocity_maximum_scaling.F90

function: `velocitymaxscalingconstructorparameters`
Description: Constructor for the `velocityMaxScaling` timescale for star formation in spheroids class which takes a parameter set as input.
Code lines: 46
Contained by: file star_formation.timescales.spheroids.velocity_maximum_scaling.F90
Modules used: input_parameters

subroutine: `velocitymaxscalingdestructor`
Description: Destructor for the `velocityMaxScaling` timescale for star formation in spheroids class.
Code lines: 10
Contained by: file star_formation.timescales.spheroids.velocity_maximum_scaling.F90
Modules used: events_hooks

function: `velocitymaxscalingtimescale`
Description: Returns the timescale (in Gyr) for star formation in the galactic spheroid of `node` in the halo scaling timescale model.
Code lines: 36

Contained by: file star_formation.timescales.spheroids.velocity_maximum_scaling.F90
Modules used: galacticus_nodes

file: statistics.Nbody.halos.mass_errors.F90
Description: Contains a module which provides a class that implements errors on dark matter halo masses in N-body simulations.
Code lines: 56

module: statistics_nbody_halo_mass_errors
Code lines: 33
Contained by: file statistics.Nbody.halos.mass_errors.F90
Modules used: galacticus_nodes
Used by: file dark_matter_-halos.spins.distributions.N-body_-errors.F90; subroutine galacticus_function_-classes_destroy; function concentrationdistributioncdmcococonstructorinternal; function concentrationdistributioncdmcococonstructorparam; file galacticus.output.analyses.concentration_vs_mass_relation.CDM.Ludlow_2016.F90; file galacticus.output.analyses.distribution_-operator.random_error.N-body_mass.F90; file galacticus.output.analyses.spin_-distribution.Bett2007.F90; file galacticus.output.analyses.weight_-operator.N-body_mass.F90; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; file merger_-trees.operators.conditional_-mass_function.F90; file merger_trees.operators.perturb_-masses.F90; function halomassfunctionevaluate; file models.likelihoods.spin_-distribution.F90; file structure_formation.halo_mass_-function.error_convolved.F90

file: statistics.Nbody.halos.mass_errors.SO_halo_finder.F90
Description: Contains a module which implements an N-body dark matter halo mass error class which implements a model for errors in spherical overdensity halo finders.
Code lines: 146
Modules used: dark_matter_halo_scales; dark_matter_profiles_dmo

interface: nbodyhalomasserrorsohalofinder
Description: Constructors for the soHaloFinder N-body halo mass error class.
Code lines: 4
Contained by: file statistics.Nbody.halos.mass_errors.SO_halo_finder.F90

function: sohalofindercorrelation
Description: Return the correlation of the masses of a pair of N-body halos.
Code lines: 17
Contained by: file statistics.Nbody.halos.mass_errors.SO_halo_finder.F90
Modules used: galacticus_nodes

subroutine: `sohalofinderdestructor`
Description: Destructor for the `soHaloFinder` N-body halo mass error class.
Code lines: 8
Contained by: file `statistics.Nbody.halos.mass_errors.SO_halo_finder.F90`

function: `sohalofindererrorfractional`
Description: Return the fractional error on the mass of an N-body halo in the power-law error model.
Code lines: 27
Contained by: file `statistics.Nbody.halos.mass_errors.SO_halo_finder.F90`
Modules used: `galacticus_nodes` `numerical_constants_math`

function: `sohalofinderinternal`
Description: Internal constructor for the `soHaloFinder` N-body halo mass error class.
Code lines: 10
Contained by: file `statistics.Nbody.halos.mass_errors.SO_halo_finder.F90`

function: `sohalofinderparameters`
Description: Constructor for the `soHaloFinder` N-body halo mass error class which takes a parameter set as input.
Code lines: 25
Contained by: file `statistics.Nbody.halos.mass_errors.SO_halo_finder.F90`
Modules used: `input_parameters`

file: `statistics.Nbody.halos.mass_errors.Trenti2010.F90`
Description: Contains a module which implements an N-body dark matter halo mass error class using the model of Trenti et al. [2010].
Code lines: 166
Modules used: `cosmology_functions`

interface: `nbodyhalomasserrortrenti2010`
Description: Constructors for the `trenti2010` N-body halo mass error class.
Code lines: 4
Contained by: file `statistics.Nbody.halos.mass_errors.Trenti2010.F90`

function: `nbodyhalomasserrortrenti2010internal`
Description: Internal constructor for the `trenti2010` N-body halo mass error class. Trenti et al. [2010] report a normalization of the fractional error in particle number of 0.15 at $N = 1000$ particles. Since this is based on comparisons of halos in simulations differing in number of particles by a factor 8 this actually overestimates the normalization by a factor $\sqrt{5/4}$. Therefore, we use a normalization of 0.135 here.
Code lines: 34
Contained by: file `statistics.Nbody.halos.mass_errors.Trenti2010.F90`
Modules used: `galacticus_error`

function: `nbodyhalomasserrortrenti2010parameters`
Description: Constructor for the `trenti2010` N-body halo mass error class which takes a parameter set as input.
Code lines: 50
Contained by: file `statistics.Nbody.halos.mass_errors.Trenti2010.F90`

Modules used: input_parameters

function: trenti2010correlation
Description: Return the correlation of the masses of a pair of N-body halos.
Code lines: 20
Contained by: file statistics.Nbody.halos.mass_errors.Trenti2010.F90
Modules used: galacticus_nodes

subroutine: trenti2010destructor
Description: Destructor for the trenti2010 N-body statistics class.
Code lines: 9
Contained by: file statistics.Nbody.halos.mass_errors.Trenti2010.F90

file: statistics.Nbody.halos.mass_errors.friends-of-friends.F90
Description: Contains a module which implements an N-body dark matter halo mass error class using a fit appropriate for friends-of-friends group finders.
Code lines: 77

interface: nbodyhalomasserrorfriendsoffriends
Description: Constructors for the friendsOfFriends N-body halo mass error class.
Code lines: 4
Contained by: file statistics.Nbody.halos.mass_errors.friends-of-friends.F90

function: nbodyhalomasserrorfriendsoffriendsinternal
Description: Internal constructor for the friendsOfFriends N-body halo mass error class.
Code lines: 15
Contained by: file statistics.Nbody.halos.mass_errors.friends-of-friends.F90

function: nbodyhalomasserrorfriendsoffriendsparameters
Description: Constructor for the friendsOfFriends N-body halo mass error class which takes a parameter set as input.
Code lines: 20
Contained by: file statistics.Nbody.halos.mass_errors.friends-of-friends.F90
Modules used: input_parameters

file: statistics.Nbody.halos.mass_errors.null.F90
Description: Contains a module which implements a null N-body dark matter halo mass error class.
Code lines: 93

interface: nbodyhalomasserrornull
Description: Constructors for the null N-body halo mass error class.
Code lines: 3
Contained by: file statistics.Nbody.halos.mass_errors.null.F90

function: nbodyhalomasserrornullparameters
Description: Constructor for the null N-body halo mass error class which takes a parameter set as input.
Code lines: 11
Contained by: file statistics.Nbody.halos.mass_errors.null.F90
Modules used: input_parameters

function: `nullcorrelation`
Description: Return the correlation of the masses of a pair of N-body halos.
Code lines: 17
Contained by: file `statistics.Nbody.halos.mass_errors.null.F90`
Modules used: `galacticus_nodes`

function: `nullerrorfractional`
Description: Return the fractional error on the mass of an N-body halo.
Code lines: 9
Contained by: file `statistics.Nbody.halos.mass_errors.null.F90`

function: `nullerrorzeroalways`
Description: Return true since errors are always zero in this model.
Code lines: 8
Contained by: file `statistics.Nbody.halos.mass_errors.null.F90`

file: `statistics.Nbody.halos.mass_errors.power_law.F90`
Description: Contains a module which implements an N-body dark matter halo mass error class in which errors are a power-law in halo mass.
Code lines: 136

interface: `nbodyhalomasserrorpowerlaw`
Description: Constructors for the `powerLaw` N-body halo mass error class.
Code lines: 4
Contained by: file `statistics.Nbody.halos.mass_errors.power_law.F90`

function: `nbodyhalomasserrorpowerlawinternal`
Description: Internal constructor for the `powerLaw` N-body halo mass error class.
Code lines: 10
Contained by: file `statistics.Nbody.halos.mass_errors.power_law.F90`

function: `nbodyhalomasserrorpowerlawparameters`
Description: Constructor for the `powerLaw` N-body halo mass error class which takes a parameter set as input.
Code lines: 45
Contained by: file `statistics.Nbody.halos.mass_errors.power_law.F90`
Modules used: `input_parameters`

function: `powerlawcorrelation`
Description: Return the correlation of the masses of a pair of N-body halos.
Code lines: 17
Contained by: file `statistics.Nbody.halos.mass_errors.power_law.F90`
Modules used: `galacticus_nodes`

function: `powerlawerrorfractional`
Description: Return the fractional error on the mass of an N-body halo in the power-law error model.
Code lines: 11
Contained by: file `statistics.Nbody.halos.mass_errors.power_law.F90`

Modules used: galacticus_nodes

file: statistics.distributions.Cauchy.F90
Description: Implementation of a 1D Cauchy distibution function.
Code lines: 125

function: cauchyconstructorinternal
Description: Constructor for "cauchy" 1D distribution function class.
Code lines: 8
Contained by: file statistics.distributions.Cauchy.F90

function: cauchyconstructorparameters
Description: Constructor for the cauchy 1D distribution function class which builds the object from a parameter set.
Code lines: 26
Contained by: file statistics.distributions.Cauchy.F90
Modules used: input_parameters

function: cauchyconstructorprobability
Description: Constructor for "cauchy" 1D distribution function class.
Code lines: 8
Contained by: file statistics.distributions.Cauchy.F90
Modules used: numerical_constants_math

function: cauchycumulative
Description: Return the cumulative probability of a Cauchy distribution.
Code lines: 9
Contained by: file statistics.distributions.Cauchy.F90
Modules used: numerical_constants_math

function: cauchydensity
Description: Return the density of a Cauchy distribution.
Code lines: 9
Contained by: file statistics.distributions.Cauchy.F90
Modules used: numerical_constants_math

function: cauchyinverse
Description: Return the inverse of a Cauchy distribution.
Code lines: 11
Contained by: file statistics.distributions.Cauchy.F90
Modules used: galacticus_error numerical_constants_math

interface: distributionfunction1dcauchy
Description: Constructors for the cauchy 1D distribution function class.
Code lines: 5
Contained by: file statistics.distributions.Cauchy.F90

file: statistics.distributions.F90
Description: Contains a module that implements a class of distributions.

Code lines: 106

module: `statistics_distributions`
Code lines: 84
Contained by: file statistics.distributions.F90
Modules used: pseudo_random
Used by: function morphologicalfractiongamamoffett2016constructorinternal; file merger_trees.constructor.builder.Cole2000.F90; file merger_trees.operators.perturb_masses.F90; file models.parameters.active.F90; file satellites.merging.virial_orbits.Jiang2014.F90; file satellites.orphans.distributions.random_isotropic.F90; file structure_formation.halo_environment.lognormal.F90; file structure_formation.halo_environment.normal.F90; program test_math_distributions

type: `distributionfunction1dlist`
Code lines: 2
Contained by: module statistics_distributions

file: `statistics.distributions.Gamma.F90`
Description: Implementation of a 1D gamma distibution function.
Code lines: 165

interface: `distributionfunction1dgamma`
Description: Constructors for the `gamma` 1D distribution function class.
Code lines: 4
Contained by: file statistics.distributions.Gamma.F90

function: `gammaconstructorinternal`
Description: Constructor for "gamma" 1D distribution function class.
Code lines: 32
Contained by: file statistics.distributions.Gamma.F90
Modules used: galacticus_error

function: `gammaconstructorparameters`
Description: Constructor for the `gamma` 1D distribution function class which builds the object from a parameter set.
Code lines: 40
Contained by: file statistics.distributions.Gamma.F90
Modules used: input_parameters

function: `gammacumulative`
Description: Return the cumulative probability of a Gamma distribution.
Code lines: 17
Contained by: file statistics.distributions.Gamma.F90
Modules used: gamma_functions

function: `gammadensity`
Description: Return the density of a Gamma distribution.
Code lines: 13
Contained by: file `statistics.distributions.Gamma.F90`
Modules used: `gamma_functions`

function: `gammainverse`
Description: Return the inverse of a Gamma distribution.
Code lines: 11
Contained by: file `statistics.distributions.Gamma.F90`
Modules used: `galacticus_error` `gamma_functions`

file: `statistics.distributions.Student-t.F90`
Description: Implementation of a 1D Student-t distribution function.
Code lines: 107

interface: `distributionfunction1dstudentt`
Description: Constructors for the `studentT` 1D distribution function class.
Code lines: 4
Contained by: file `statistics.distributions.Student-t.F90`

function: `studenttconstructorinternal`
Description: Constructor for "studentT" 1D distribution function class.
Code lines: 8
Contained by: file `statistics.distributions.Student-t.F90`

function: `studenttconstructorparameters`
Description: Constructor for the `studentT` 1D distribution function class which builds the object from a parameter set.
Code lines: 19
Contained by: file `statistics.distributions.Student-t.F90`
Modules used: `input_parameters`

function: `studenttcumulative`
Description: Return the cumulative probability of a Student-t distribution.
Code lines: 9
Contained by: file `statistics.distributions.Student-t.F90`
Modules used: `fgsl`

function: `studenttdensity`
Description: Return the density of a Student-t distribution.
Code lines: 9
Contained by: file `statistics.distributions.Student-t.F90`
Modules used: `fgsl`

function: `studenttinverse`
Description: Return the cumulative probability of a Student-t distribution.
Code lines: 11

Contained by: file statistics.distributions.Student-t.F90
Modules used: fgsl galacticus_error

file: statistics.distributions.Voight.F90
Description: Implementation of a peak-background split density 1D distibution function.
Code lines: 174

interface: distributionfunction1dvoight
Description: Constructors for the voight 1D distribution function class.
Code lines: 4
Contained by: file statistics.distributions.Voight.F90

function: voightconstructorinternal
Description: Constructor for "voight" 1D distribution function class.
Code lines: 28
Contained by: file statistics.distributions.Voight.F90

function: voightconstructorparameters
Description: Constructor for the voight 1D distribution function class which builds the object from a parameter set.
Code lines: 49
Contained by: file statistics.distributions.Voight.F90
Modules used: input_parameters

function: voightcumulative
Description: Return the cumulative probability of a Voight distribution.
Code lines: 26
Contained by: file statistics.distributions.Voight.F90
Modules used: error_functions hypergeometric_functions numerical_constants_math

function: voightdensity
Description: Return the density of a Voight distribution.
Code lines: 22
Contained by: file statistics.distributions.Voight.F90
Modules used: error_functions numerical_constants_math

file: statistics.distributions.beta.F90
Description: Implementation of a beta density 1D distibution function.
Code lines: 173

function: betaconstructorinternal
Description: Constructor for "beta" 1D distribution function class.
Code lines: 8
Contained by: file statistics.distributions.beta.F90

function: betaconstructorparameters
Description: Constructor for the beta 1D distribution function class which builds the object from a parameter set.
Code lines: 26

Contained by: file statistics.distributions.beta.F90
Modules used: input_parameters

function: betacumulative
Description: Return the cumulative probability of a beta distribution.
Code lines: 15
Contained by: file statistics.distributions.beta.F90
Modules used: beta_functions

function: betadensity
Description: Return the density of a beta distribution.
Code lines: 13
Contained by: file statistics.distributions.beta.F90
Modules used: beta_functions

function: betainverse
Description: Return the inverse of a beta distribution.
Code lines: 24
Contained by: file statistics.distributions.beta.F90
Modules used: galacticus_error root_finder

function: betamaximum
Description: Return the minimum value of a beta distribution.
Code lines: 8
Contained by: file statistics.distributions.beta.F90

function: betaminimum
Description: Return the minimum value of a beta distribution.
Code lines: 8
Contained by: file statistics.distributions.beta.F90

function: betaroot
Description: Root function used in finding the inverse of the beta distribution.
Code lines: 7
Contained by: file statistics.distributions.beta.F90

interface: distributionfunction1dbeta
Description: Constructors for the beta 1D distribution function class.
Code lines: 4
Contained by: file statistics.distributions.beta.F90

file: statistics.distributions.discrete.F90
Description: Contains a module that implements a class of discrete distributions.
Code lines: 106

module: statistics_distributions_discrete
Code lines: 84
Contained by: file statistics.distributions.discrete.F90
Modules used: pseudo_random
Used by: function localgroupmassfunctionloglikelihood

file: `statistics.distributions.discrete.binomial.F90`
Description: Implementation of a binomial 1D discrete distibution function.
Code lines: 170

function: `binomialconstructorinternal`
Description: Constructor for "binomial" 1D distribution function class.
Code lines: 23
Contained by: file `statistics.distributions.discrete.binomial.F90`
Modules used: `galacticus_error`

function: `binomialconstructorparameters`
Description: Constructor for the `binomial` 1D discrete distribution function class which builds the object from a parameter set.
Code lines: 27
Contained by: file `statistics.distributions.discrete.binomial.F90`
Modules used: `input_parameters`

function: `binomialcumulative`
Description: Return the cumulative probability of a binomial discrete distribution.
Code lines: 10
Contained by: file `statistics.distributions.discrete.binomial.F90`
Modules used: `galacticus_error`

function: `binomialinverse`
Description: Return the inverse of a binomial discrete distribution.
Code lines: 13
Contained by: file `statistics.distributions.discrete.binomial.F90`
Modules used: `galacticus_error`

function: `binomialmass`
Description: Return the mass of a binomial discrete distribution.
Code lines: 11
Contained by: file `statistics.distributions.discrete.binomial.F90`
Modules used: `factorials` `galacticus_error`

function: `binomialmasslogarithmic`
Description: Return the logarithmic mass of a binomial discrete distribution.
Code lines: 8
Contained by: file `statistics.distributions.discrete.binomial.F90`

function: `binomialmaximum`
Description: Return the maximum possible value in a binomial discrete distribution.
Code lines: 8
Contained by: file `statistics.distributions.discrete.binomial.F90`

function: `binomialminimum`
Description: Return the minimum possible value in a binomial discrete distribution.
Code lines: 8
Contained by: file `statistics.distributions.discrete.binomial.F90`

interface: `distributionfunctiondiscrete1dbinomial`
Description: Constructors for the `binomial` 1D discrete distribution function class.
Code lines: 4
Contained by: file `statistics.distributions.discrete.binomial.F90`

file: `statistics.distributions.discrete.negative_binomial.F90`
Description: Implementation of a negative binomial 1D discrete distibution function.
Code lines: 158

interface: `distributionfunctiondiscrete1dnegativebinomial`
Description: Constructors for the `negativeBinomial` 1D discrete distribution function class.
Code lines: 4
Contained by: file `statistics.distributions.discrete.negative_binomial.F90`

function: `negativebinomialconstructorinternal`
Description: Constructor for "negativeBinomial" 1D distribution function class.
Code lines: 12
Contained by: file `statistics.distributions.discrete.negative_binomial.F90`
Modules used: `galacticus_error`

function: `negativebinomialconstructorparameters`
Description: Constructor for the `negativeBinomial` 1D discrete distribution function class which builds the object from a parameter set.
Code lines: 26
Contained by: file `statistics.distributions.discrete.negative_binomial.F90`
Modules used: `input_parameters`

function: `negativebinomialcumulative`
Description: Return the cumulative probability of a negativeBinomial discrete distribution.
Code lines: 11
Contained by: file `statistics.distributions.discrete.negative_binomial.F90`
Modules used: `galacticus_error`

function: `negativebinomialinverse`
Description: Return the inverse of a negative binomial discrete distribution.
Code lines: 11
Contained by: file `statistics.distributions.discrete.negative_binomial.F90`
Modules used: `galacticus_error`

function: `negativebinomialmass`
Description: Return the mass of a negative binomial discrete distribution.
Code lines: 11
Contained by: file `statistics.distributions.discrete.negative_binomial.F90`
Modules used: `galacticus_error` `gamma_functions`

function: `negativebinomialmasslogarithmic`
Description: Return the logarithmic mass of a negative binomial discrete distribution.
Code lines: 11

Contained by: file statistics.distributions.discrete.negative_binomial.F90
Modules used: galacticus_error gamma_functions

function: negativebinomialmaximum
Description: Return the maximum possible value in a negative binomial discrete distribution.
Code lines: 8
Contained by: file statistics.distributions.discrete.negative_binomial.F90

function: negativebinomialminimum
Description: Return the minimum possible value in a negative binomial discrete distribution.
Code lines: 8
Contained by: file statistics.distributions.discrete.negative_binomial.F90

file: statistics.distributions.lognormal.F90
Description: Implementation of a normal 1D distibution function.
Code lines: 158

interface: distributionfunction1dlognormal
Description: Constructors for the normal 1D distribution function class.
Code lines: 4
Contained by: file statistics.distributions.lognormal.F90

function: lognormalconstructorinternal
Description: Constructor for "normal" 1D distribution function class.
Code lines: 24
Contained by: file statistics.distributions.lognormal.F90

function: lognormalconstructorparameters
Description: Constructor for the normal 1D distribution function class which builds the object from a parameter set.
Code lines: 40
Contained by: file statistics.distributions.lognormal.F90
Modules used: input_parameters

function: lognormalcumulative
Description: Return the cumulative probability of a normal distribution.
Code lines: 8
Contained by: file statistics.distributions.lognormal.F90

function: lognormaldensity
Description: Return the density of a normal distribution.
Code lines: 8
Contained by: file statistics.distributions.lognormal.F90

function: lognormalinverse
Description: Return the inverse of a normal distribution.
Code lines: 8
Contained by: file statistics.distributions.lognormal.F90

function: `lognormalmaximum`
Description: Return the maximum possible value of a uniform distribution.
Code lines: 7
Contained by: file `statistics.distributions.lognormal.F90`

function: `lognormalminimum`
Description: Return the minimum possible value of a uniform distribution.
Code lines: 7
Contained by: file `statistics.distributions.lognormal.F90`

file: `statistics.distributions.loguniform.F90`
Description: Implementation of a 1D distibution function which is uniform in the logarithm of the variable.
Code lines: 121

interface: `distributionfunction1dloguniform`
Description: Constructors for the `logUniform` 1D distribution function class.
Code lines: 4
Contained by: file `statistics.distributions.loguniform.F90`

function: `loguniformconstructorinternal`
Description: Constructor for "logUniform" 1D distribution function class.
Code lines: 8
Contained by: file `statistics.distributions.loguniform.F90`

function: `loguniformconstructorparameters`
Description: Constructor for the `logUniform` 1D distribution function class which builds the object from a parameter set.
Code lines: 26
Contained by: file `statistics.distributions.loguniform.F90`
Modules used: `input_parameters`

function: `loguniformcumulative`
Description: Return the cumulative probability of a uniform distribution.
Code lines: 14
Contained by: file `statistics.distributions.loguniform.F90`

function: `loguniformdensity`
Description: Return the density of a uniform distribution.
Code lines: 12
Contained by: file `statistics.distributions.loguniform.F90`

function: `loguniforminverse`
Description: Return the inverse of a uniform distribution.
Code lines: 10
Contained by: file `statistics.distributions.loguniform.F90`
Modules used: `galacticus_error`

file: `statistics.distributions.negative_exponential.F90`
Description: Implementation of a negative exponential density 1D distibution function.
Code lines: 119

interface: `distributionfunction1dnegativeexponential`
Description: Constructors for the `negativeExponential` 1D distribution function class.
Code lines: 4
Contained by: file `statistics.distributions.negative_exponential.F90`

function: `negativeexponentialconstructorinternal`
Description: Constructor for "negativeExponential" 1D distribution function class.
Code lines: 8
Contained by: file `statistics.distributions.negative_exponential.F90`

function: `negativeexponentialconstructorparameters`
Description: Constructor for the `negativeExponential` 1D distribution function class which builds the object from a parameter set.
Code lines: 19
Contained by: file `statistics.distributions.negative_exponential.F90`
Modules used: `input_parameters`

function: `negativeexponentialcumulative`
Description: Return the cumulative probability of a negative exponential distribution.
Code lines: 12
Contained by: file `statistics.distributions.negative_exponential.F90`

function: `negativeexponentialdensity`
Description: Return the density of a negative exponential distribution.
Code lines: 12
Contained by: file `statistics.distributions.negative_exponential.F90`

function: `negativeexponentialinverse`
Description: Return the inverse of a negative exponential distribution.
Code lines: 17
Contained by: file `statistics.distributions.negative_exponential.F90`
Modules used: `galacticus_error`

file: `statistics.distributions.normal.F90`
Description: Implementation of a normal 1D distibution function.
Code lines: 272

interface: `distributionfunction1dnormal`
Description: Constructors for the `normal` 1D distribution function class.
Code lines: 4
Contained by: file `statistics.distributions.normal.F90`

function: `normalconstructorinternal`
Description: Constructor for "normal" 1D distribution function class.

Code lines: 27
Contained by: file statistics.distributions.normal.F90
Modules used: error_functions

function: normalconstructorparameters
Description: Constructor for the normal 1D distribution function class which builds the object from a parameter set.
Code lines: 48
Contained by: file statistics.distributions.normal.F90
Modules used: input_parameters

function: normalcumulative
Description: Return the cumulative probability of a normal distribution.
Code lines: 15
Contained by: file statistics.distributions.normal.F90
Modules used: error_functions

function: normaldensity
Description: Return the density of a normal distribution.
Code lines: 13
Contained by: file statistics.distributions.normal.F90
Modules used: numerical_constants_math

function: normalinverse
Description: Return the inverse of a normal distribution.
Code lines: 10
Contained by: file statistics.distributions.normal.F90
Modules used: galacticus_error

function: normalmaximum
Description: Return the maximum possible value of a uniform distribution.
Code lines: 13
Contained by: file statistics.distributions.normal.F90
Modules used: galacticus_error

function: normalminimum
Description: Return the minimum possible value of a uniform distribution.
Code lines: 13
Contained by: file statistics.distributions.normal.F90
Modules used: galacticus_error

function: normalpolynomialevaluate

Description: Evaluates a polynomial based on the implementation by John Burkardt. For sanity's sake, the value of N indicates the *number* of coefficients, or more precisely, the *order* of the polynomial, rather than the *degree* of the polynomial. The two quantities differ by 1, but cause a great deal of confusion. Given `n` and `a`, the form of the polynomial is:

$$p(x) = a(1) + a(2) * x + \ldots + a(n-1) * x^{n-2} + a(n) * x^{n-1}. \tag{18.51}$$

Code lines: 19

Contained by: file `statistics.distributions.normal.F90`

function: `normalstandardinverse`

Description: Evaluates the inverse of the standard normal cumulative distribution function. Based on the Fortran90 version by John Burkardt (itself based on the original Fortran 77 version by Michael Wichura), using the alogorithm of Wichura [1988].
Code lines: 52
Contained by: file statistics.distributions.normal.F90
Modules used: galacticus_error

file: statistics.distributions.peak_background.F90
Description: Implementation of a peak-background split density 1D distibution function.
Code lines: 191
Modules used: tables

interface: distributionfunction1dpeakbackground
Description: Constructors for the peakBackground 1D distribution function class.
Code lines: 4
Contained by: file statistics.distributions.peak_background.F90

function: peakbackgroundconstructorinternal
Description: Constructor for "peakBackground" 1D distribution function class.
Code lines: 33
Contained by: file statistics.distributions.peak_background.F90
Modules used: error_functions

function: cdfintegrand
Description: The integrand for the cumulative distribution function.
Code lines: 7
Contained by: function peakbackgroundconstructorinternal

function: peakbackgroundconstructorparameters
Description: Constructor for the peakBackground 1D distribution function class which builds the object from a parameter set.
Code lines: 26
Contained by: file statistics.distributions.peak_background.F90
Modules used: input_parameters

function: peakbackgroundcumulative
Description: Return the cumulative probability of a normal distribution.
Code lines: 14
Contained by: file statistics.distributions.peak_background.F90

function: peakbackgrounddensity
Description: Return the density of a normal distribution.
Code lines: 13
Contained by: file statistics.distributions.peak_background.F90
Modules used: numerical_constants_math

subroutine: peakbackgrounddestructor
Description: Destructor for "peakBackground" 1D distribution function class.
Code lines: 11

Contained by: file statistics.distributions.peak_background.F90

function: peakbackgroundinverse
Description: Return the inverse of a normal distribution.
Code lines: 10
Contained by: file statistics.distributions.peak_background.F90
Modules used: galacticus_error

function: peakbackgroundmaximum
Description: Return the maximum possible value of a peak-background split distribution.
Code lines: 10
Contained by: file statistics.distributions.peak_background.F90
Modules used: galacticus_error

function: peakbackgroundminimum
Description: Return the minimum possible value of a peak-background split distribution.
Code lines: 10
Contained by: file statistics.distributions.peak_background.F90
Modules used: galacticus_error

file: statistics.distributions.uniform.F90
Description: Implementation of a uniform 1D distibution function.
Code lines: 141

interface: distributionfunction1duniform
Description: Constructors for the uniform 1D distribution function class.
Code lines: 4
Contained by: file statistics.distributions.uniform.F90

function: uniformconstructorinternal
Description: Constructor for "uniform" 1D distribution function class.
Code lines: 8
Contained by: file statistics.distributions.uniform.F90

function: uniformconstructorparameters
Description: Constructor for the uniform 1D distribution function class which builds the object from a parameter set.
Code lines: 26
Contained by: file statistics.distributions.uniform.F90
Modules used: input_parameters

function: uniformcumulative
Description: Return the cumulative probability of a uniform distribution.
Code lines: 14
Contained by: file statistics.distributions.uniform.F90

function: uniformdensity
Description: Return the density of a uniform distribution.
Code lines: 12
Contained by: file statistics.distributions.uniform.F90

function: `uniforminverse`
Description: Return the inverse of a uniform distribution.
Code lines: 10
Contained by: file `statistics.distributions.uniform.F90`
Modules used: `galacticus_error`

function: `uniformmaximum`
Description: Return the maximum possible value of a uniform distribution.
Code lines: 7
Contained by: file `statistics.distributions.uniform.F90`

function: `uniformminimum`
Description: Return the minimum possible value of a uniform distribution.
Code lines: 7
Contained by: file `statistics.distributions.uniform.F90`

file: `statistics.mass_function.incompleteness.F90`
Description: Contains a module that provides an object that implements incompleteness calculations for observed mass functions.
Code lines: 39

module: `mass_function_incompletenesses`
Description: Provides a class that implements incompleteness calculations for observed mass functions.
Code lines: 17
Contained by: file `statistics.mass_function.incompleteness.F90`
Used by: subroutine `galacticus_function_classes_destroy`; subroutine `galacticus_state_retrieve`; subroutine `galacticus_state_store`; function `massfunctionevaluate`; file `tasks.conditional_mass_function.F90`

file: `statistics.mass_function.incompleteness.complete.F90`
Description: Implements calculations of incompleteness assuming a complete sample.
Code lines: 60

function: `completecompleteness`
Description: Return the completeness.
Code lines: 9
Contained by: file `statistics.mass_function.incompleteness.complete.F90`

function: `completeconstructorparameters`
Description: Constructor for the `complete` incompleteness class which takes a parameter set as input.
Code lines: 10
Contained by: file `statistics.mass_function.incompleteness.complete.F90`
Modules used: `input_parameters`

interface: `massfunctionincompletenesscomplete`
Description: Constructors for the `complete` incompleteness class.

Code lines: 3
Contained by: file statistics.mass_function.incompleteness.complete.F90

file: statistics.mass_function.incompleteness.surface_brightness.F90
Description: Implements calculations of incompleteness assuming a normal distribution of surface brightnesses.
Code lines: 111

interface: massfunctionincompletenesssurfacebrightness
Description: Constructors for the "surface brightness" incompleteness class.
Code lines: 4
Contained by: file statistics.mass_function.incompleteness.surface_brightness.F90

function: surfacebrightnesscompleteness
Description: Return the completeness.
Code lines: 12
Contained by: file statistics.mass_function.incompleteness.surface_brightness.F90
Modules used: error_functions

function: surfacebrightnessconstructorinternal
Description: Internal constructor for the "surface brightness" incompleteness class.
Code lines: 8
Contained by: file statistics.mass_function.incompleteness.surface_brightness.F90

function: surfacebrightnessconstructorparameters
Description: Constructor for the surfaceBrightness incompleteness class which takes a parameter set as input.
Code lines: 46
Contained by: file statistics.mass_function.incompleteness.surface_brightness.F90
Modules used: input_parameters

file: statistics.points.correlation.F90
Description: Contains a module which computes correlation statistics from point distributions.
Code lines: 232

module: statistics_points_correlations
Description: Compute correlation statistics from point distributions.
Code lines: 210
Contained by: file statistics.points.correlation.F90
Used by: subroutine catalogprojectedcorrelationfunctionperform

subroutine: statistics_points_correlation
Description: Compute the correlation function from a set of points.
Code lines: 201
Contained by: module statistics_points_correlations
Modules used: galacticus_display, galacticus_error, kind_numbers, memory_management, nearest_neighbors, numerical_ranges

vectors

file: statistics.points.power_spectrum.F90
Description: Contains a module which computes power spectra from point distributions.
Code lines: 122

module: statistics_points_power_spectra
Description: Compute power spectra from point distributions.
Code lines: 100
Contained by: file statistics.points.power_spectrum.F90

subroutine: statistics_points_power_spectrum
Description: Compute the power spectrum from a set of points in a periodic cube.
Code lines: 91
Contained by: module statistics_points_power_spectra
Modules used: fftw3, galacticus_display, galacticus_error, iso_c_binding, iso_varying_string, memory_management, meshes, numerical_constants_math, numerical_ranges, string_handling

file: stellar_astrophysics.F90
Description: Contains a module which implements a class for calculations of stellar astrophysics.
Code lines: 59

module: stellar_astrophysics
Description: Implements a class for calculations of stellar astrophysics.
Code lines: 37
Contained by: file stellar_astrophysics.F90
Modules used: iso_varying_string
Used by: program benchmark_stellar_populations_luminosities, subroutine galacticus_function_classes_destroy, subroutine galacticus_state_retrieve, subroutine galacticus_state_store, file stellar_astrophysics.feedback.standard.F90, file stellar_astrophysics.supernovae_PopulationIII.HegerWoosley2002.F90, file stellar_astrophysics.supernovae_type_Ia.Nagashima2005.F90, file stellar_populations.standard.F90, program test_stellar_populations, program test_stellar_populations_luminosities

file: stellar_astrophysics.feedback.F90
Description: Contains a module which implements a class that performs calculations of stellar feedback.
Code lines: 46

module: stellar_feedback
Description: Implements a class that performs calculations of stellar feedback.
Code lines: 24
Contained by: file stellar_astrophysics.feedback.F90

Used by: program benchmark_stellar_populations_luminosities; subroutine galacticus_function_classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; subroutine node_component_disk_very_simple_rate_compute; subroutine node_component_disk_very_simple_rates; subroutine node_component_spheroid_very_simple_rate_compute; subroutine node_component_spheroid_very_simple_rates; function creasey2012outflowrate; function fixedoutflowrate; function haloscalingconstructorinternal; function powerlawoutflowrate; function vlctymxsclngconstructorinternal; function fixedoutflowrate; function powerlawoutflowrate; function vlctymxsclngconstructorinternal; function superwindoutflowrate; function superwindoutflowrate; subroutine instantaneousrates; subroutine noninstantaneousscales; file stellar_populations.standard.F90; program test_stellar_populations; program test_stellar_populations_luminosities

file: `stellar_astrophysics.feedback.standard.F90`

Description: Implements a stellar feedback class which performs a simple calculation of energy feedback from stellar populations.

Code lines: 177

Modules used: stellar_astrophysics; stellar_astrophysics_winds; supernovae_population_iii; supernovae_type_ia

function: `standardconstructorinternal`

Description: Constructor for the `standard` stellar feedback class which takes a parameter list as input.

Code lines: 12

Contained by: file stellar_astrophysics.feedback.standard.F90

function: `standardconstructorparameters`

Description: Constructor for the `standard` stellar feedback class which takes a parameter list as input.

Code lines: 44

Contained by: file stellar_astrophysics.feedback.standard.F90

Modules used: input_parameters; numerical_constants_astronomical; numerical_constants_prefixes; numerical_constants_units

subroutine: `standarddestructor`

Description: Destructor for the `standard` stellar feedback class.

Code lines: 10

Contained by: file stellar_astrophysics.feedback.standard.F90

function: `standardenergyinputcumulative`

Description: Compute the cumulative energy input from a star of given `initialMass`, `age` and `metallicity`.

Code lines: 40

Contained by: file stellar_astrophysics.feedback.standard.F90

Modules used: `fgsl`; numerical_constants_astronomical; numerical_integration

function: `standardwindenergyintegrand`
Description: Integrand used in evaluating cumulative energy input from winds.
Code lines: 7
Contained by: file `stellar_astrophysics.feedback.standard.F90`

interface: `stellarfeedbackstandard`
Description: Constructors for the `standard` stellar feedback class.
Code lines: 4
Contained by: file `stellar_astrophysics.feedback.standard.F90`

file: `stellar_astrophysics.file.F90`
Description: Implements a stellar astrophysics class in which the stellar properties are read from file and interpolated.
Code lines: 311
Modules used: `numerical_interpolation_2d_irregular`

function: `fileconstructorinternal`
Description: Internal constructor for the `file` stellar astrophysics class.
Code lines: 18
Contained by: file `stellar_astrophysics.file.F90`
Modules used: `atomic_data` `memory_management`

function: `fileconstructorparameters`
Description: Constructor for the `file` stellar astrophysics class which takes a parameter list as input.
Code lines: 20
Contained by: file `stellar_astrophysics.file.F90`
Modules used: `galacticus_paths` `input_parameters`

function: `filelifetime`
Description: Return the lifetime of a star (in Gyr) given an `massInitial` and `metallicity`.
Code lines: 9
Contained by: file `stellar_astrophysics.file.F90`

function: `filemassejected`
Description: Return the mass ejected during the lifetime of a star of given `massInitial` and `metallicity`.
Code lines: 9
Contained by: file `stellar_astrophysics.file.F90`

function: `filemassinitial`
Description: Return the initial mass of a star of given `lifetime` and `metallicity`.
Code lines: 9
Contained by: file `stellar_astrophysics.file.F90`

function: `filemassyield`
Description: Return the mass of metals yielded by a star of given `massInitial` and `metallicity`.
Code lines: 18
Contained by: file `stellar_astrophysics.file.F90`

subroutine: `fileread`

Description: Read stellar astrophysics data. This is not done during object construction since it can be slow—we only perform the read if the data is actually needed.
Code lines: 149
Contained by: file `stellar_astrophysics.file.F90`
Modules used: `atomic_data`, `fox_dom`, `galacticus_error`, `io_xml`, `iso_varying_string`, `memory_management`

interface: `stellarastrophysicsfile`

Description: Constructors for the `file` stellar astrophysics class.
Code lines: 4
Contained by: file `stellar_astrophysics.file.F90`

file: `stellar_astrophysics.supernovae_PopulationIII.F90`

Description: Contains a module which implements a class for calculations of Population III supernovae.
Code lines: 40

module: `supernovae_population_iii`

Description: Implements a class for calculations of Population III supernovae.
Code lines: 18
Contained by: file `stellar_astrophysics.supernovae_PopulationIII.F90`
Used by: program `benchmark_stellar_populations_luminosities`, subroutine `galacticus_function_classes_destroy`, subroutine `galacticus_state_retrieve`, subroutine `galacticus_state_store`, file `stellar_astrophysics.feedback.standard.F90`, program `test_stellar_populations`, program `test_stellar_populations_luminosities`

file: `stellar_astrophysics.supernovae_PopulationIII.HegerWoosley2002.F90`

Description: Implements a Population III supernovae class based on Heger and Woosley [2002].
Code lines: 136
Modules used: `fgsl`, `stellar_astrophysics`

function: `hegerwoosley2002constructorinternal`

Description: Internal constructor for the `hegerWoosley2002` Population III supernovae class.
Code lines: 34
Contained by: file `stellar_astrophysics.supernovae_PopulationIII.HegerWoosley2002.F90`
Modules used: `fox_dom`, `galacticus_error`, `galacticus_paths`, `io_xml`, `iso_varying_string`, `numerical_constants_astronomical`

function: `hegerwoosley2002constructorparameters`

Description: Constructor for the `hegerWoosley2002` Population III supernovae class which takes a parameter list as input.
Code lines: 13
Contained by: file `stellar_astrophysics.supernovae_PopulationIII.HegerWoosley2002.F90`
Modules used: `input_parameters`

subroutine: `hegerwoosley2002destructor`
Description: Destructor for the `hegerWoosley2002` Population III supernovae class.
Code lines: 7
Contained by: file `stellar_astrophysics.supernovae_PopulationIII.HegerWoosley2002.F90`

function: `hegerwoosley2002energycumulative`
Description: Compute the cumulative energy input from Population III star pair instability supernovae using the results of Heger and Woosley [2002].
Code lines: 26
Contained by: file `stellar_astrophysics.supernovae_PopulationIII.HegerWoosley2002.F90`
Modules used: `numerical_interpolation`

interface: `supernovaepopulationiiihegerwoosley2002`
Description: Constructors for the `hegerWoosley2002` Population III supernovae class.
Code lines: 4
Contained by: file `stellar_astrophysics.supernovae_PopulationIII.HegerWoosley2002.F90`

file: `stellar_astrophysics.supernovae_type_Ia.F90`
Description: Contains a module which implements a class for calculations of Type Ia supernovae.
Code lines: 47

module: `supernovae_type_ia`
Description: Implements a class for calculations of Type Ia supernovae.
Code lines: 25
Contained by: file `stellar_astrophysics.supernovae_type_Ia.F90`
Used by:
- program `benchmark_stellar_populations_luminosities`
- subroutine `galacticus_function_classes_destroy`
- subroutine `galacticus_state_retrieve`
- subroutine `galacticus_state_store`
- file `stellar_astrophysics.feedback.standard.F90`
- file `stellar_populations.standard.F90`
- program `test_stellar_populations`
- program `test_stellar_populations_luminosities`

file: `stellar_astrophysics.supernovae_type_Ia.Nagashima2005.F90`
Description: Implements a supernovae type Ia class based on Nagashima et al. [2005].
Code lines: 168
Modules used: `stellar_astrophysics`

function: `nagashima2005constructorinternal`
Description: Internal constructor for the `nagashima2005` supernovae type Ia class.
Code lines: 43
Contained by: file `stellar_astrophysics.supernovae_type_Ia.Nagashima2005.F90`
Modules used:
- `atomic_data`
- `fox_dom`
- `galacticus_error`
- `galacticus_paths`
- `io_xml`
- `memory_management`

function: `nagashima2005constructorparameters`

Description: Constructor for the `nagashima2005` supernovae type Ia class which takes a parameter list as input.
Code lines: 13
Contained by: file `stellar_astrophysics.supernovae_type_Ia.Nagashima2005.F90`
Modules used: `input_parameters`

subroutine: `nagashima2005destructor`
Description: Destructor for the `nagashima2005` supernovae type Ia class.
Code lines: 7
Contained by: file `stellar_astrophysics.supernovae_type_Ia.Nagashima2005.F90`

function: `nagashima2005number`
Description: Compute the cumulative number of Type Ia supernovae originating per unit mass of stars that form with given `initialMass` and `metallicity` after a time `age`. The calculation is based on that of Nagashima et al. [2005]. The number returned here assumes a distribution of binary mass ratios and so only makes sense once it is integrated over an initial mass function.
Code lines: 29
Contained by: file `stellar_astrophysics.supernovae_type_Ia.Nagashima2005.F90`

function: `nagashima2005yield`
Description: Compute the cumulative yield from Type Ia supernovae originating per unit mass of stars that form with given `initialMass` and `metallicity` after a time `age`. The calculation is based on the Type Ia rate calculation of Nagashima et al. [2005] and the Type Ia yields from Nomoto et al. [1997]. The number returned here assumes a distribution of binary mass ratios and so only makes sense once it is integrated over an initial mass function.
Code lines: 21
Contained by: file `stellar_astrophysics.supernovae_type_Ia.Nagashima2005.F90`

interface: `supernovaetypeianagashima2005`
Description: Constructors for the `nagashima2005` supernovae type Ia class.
Code lines: 4
Contained by: file `stellar_astrophysics.supernovae_type_Ia.Nagashima2005.F90`

file: `stellar_astrophysics.tracks.F90`
Description: Contains a module which implements a class for calculation of stellar tracks.
Code lines: 46

module: `stellar_astrophysics_tracks`
Description: Implements a class for stellar tracks.
Code lines: 24
Contained by: file `stellar_astrophysics.tracks.F90`
Used by:
- program `benchmark_stellar_populations_luminosities`
- subroutine `galacticus_state_retrieve`
- file `stellar_astrophysics.winds.Leitherer1992.F90`
- program `test_stellar_populations_luminosities`
- subroutine `galacticus_function_classes_destroy`
- subroutine `galacticus_state_store`
- program `test_stellar_populations`

file: `stellar_astrophysics.tracks.file.F90`
Description: Implements a stellar tracks class in which the tracks are read from file and interpolated.
Code lines: 367
Modules used: `fgsl`

function: `fileconstructorinternal`
Description: Internal constructor for the `file` stellar tracks class.
Code lines: 109
Contained by: file `stellar_astrophysics.tracks.file.F90`
Modules used: `galacticus_error`, `io_hdf5`, `iso_varying_string`, `memory_management`, `string_handling`

function: `fileconstructorparameters`
Description: Constructor for the `file` stellar tracks class which takes a parameter list as input.
Code lines: 20
Contained by: file `stellar_astrophysics.tracks.file.F90`
Modules used: `galacticus_paths`, `input_parameters`

function: `fileinterpolate`
Description: Using precomputed factors, interpolate in metallicity, mass and age in the given `stellarTracks`.
Code lines: 28
Contained by: file `stellar_astrophysics.tracks.file.F90`
Modules used: `iso_c_binding`

subroutine: `fileinterpolationcompute`
Description: Get interpolating factors for stellar tracks.
Code lines: 82
Contained by: file `stellar_astrophysics.tracks.file.F90`
Modules used: `iso_c_binding`, `numerical_interpolation`

function: `fileluminosity`
Description: Return the bolometric luminosity (in $L_\odot$) for a star of given `initialMass`, `metallicity` and `age`.
Code lines: 23
Contained by: file `stellar_astrophysics.tracks.file.F90`
Modules used: `iso_c_binding`

function: `filetemperatureeffective`
Description: Return the effective temperature (in Kelvin) for a star of given `initialMass`, `metallicity` and `age`.
Code lines: 23
Contained by: file `stellar_astrophysics.tracks.file.F90`
Modules used: `iso_c_binding`

interface: `stellartracksfile`
Description: Constructors for the `file` stellar tracks class.

Code lines: 4
Contained by: file stellar_astrophysics.tracks.file.F90

file: stellar_astrophysics.winds.F90
Description: Contains a module which implements a class for calculations of stellar winds.
Code lines: 46

module: stellar_astrophysics_winds
Description: Implements a class for calculations of stellar winds.
Code lines: 24
Contained by: file stellar_astrophysics.winds.F90
Used by: program benchmark_stellar_populations_luminosities; subroutine galacticus_function_classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; file stellar_astrophysics.feedback.standard.F90; program test_stellar_populations; program test_stellar_populations_luminosities

file: stellar_astrophysics.winds.Leitherer1992.F90
Description: Implements a stellar winds class based on Leitherer et al. [1992].
Code lines: 123
Modules used: numerical_constants_astronomical; stellar_astrophysics_tracks

function: leitherer1992constructorinternal
Description: Internal constructor for the leitherer1992 stellar winds class.
Code lines: 8
Contained by: file stellar_astrophysics.winds.Leitherer1992.F90

function: leitherer1992constructorparameters
Description: Constructor for the leitherer1992 stellar winds class which takes a parameter list as input.
Code lines: 13
Contained by: file stellar_astrophysics.winds.Leitherer1992.F90
Modules used: input_parameters

subroutine: leitherer1992destructor
Description: Destructor for the leitherer1992 stellar winds class.
Code lines: 7
Contained by: file stellar_astrophysics.winds.Leitherer1992.F90

function: leitherer1992ratemassloss
Description: Compute the mass loss rate (in $M_\odot$/Gyr) from a star of given initialMass, age and metallicity using the fitting formula of Leitherer et al. [1992].
Code lines: 19
Contained by: file stellar_astrophysics.winds.Leitherer1992.F90

function: leitherer1992velocityterminal
Description: Compute the terminal velocity (in km/s) from a star of given initialMass, age and metallicity using the fitting formula of Leitherer et al. [1992].
Code lines: 19

Contained by: file stellar_astrophysics.winds.Leitherer1992.F90

interface: stellarwindsleitherer1992
Description: Constructors for the leitherer1992 stellar winds class.
Code lines: 4
Contained by: file stellar_astrophysics.winds.Leitherer1992.F90

file: stellar_populations.F90
Description: Contains a module which implements a class for stellar populations.
Code lines: 114

module: stellar_populations
Description: Implements a class for stellar populations.
Code lines: 92
Contained by: file stellar_populations.F90
Modules used: abundances_structure, hashes, stellar_population_spectra
Used by: program benchmark_stellar_populations_luminosities; subroutine galacticus_function_classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; function sedfitevaluate; subroutine stellar_luminosities_set; function intergalacticbackgroundinternalupdate; function stellar_population_luminosity; subroutine stellar_population_luminosity_tabulate; subroutine stellar_population_luminosity_track; subroutine instantaneousrates; subroutine noninstantaneousrates; module stellar_population_selectors; function diskspheroidconstructorparameters; function fixedconstructorparameters; program test_stellar_populations; program test_stellar_populations_luminosities

subroutine: stellarpopulationuniqueidassign
Description: Assign a unique ID to a stellar population. Populations are distinguished based on the hash of their descriptor.
Code lines: 21
Contained by: module stellar_populations
Modules used: galacticus_error

file: stellar_populations.initial_mass_functions.BPASS.F90
Description: Implements a stellar initial mass function class used by the BPASS library.
Code lines: 70

function: bpassconstructorinternal
Description: Internal constructor for the bpass initial mass function.
Code lines: 7
Contained by: file stellar_populations.initial_mass_functions.BPASS.F90

function: bpassconstructorparameters

Description: Constructor for the `bpass` initial mass function class which takes a parameter list as input.
Code lines: 10
Contained by: file `stellar_populations.initial_mass_functions.BPASS.F90`
Modules used: `input_parameters`

function: `bpasslabel`
Description: Return a label for this IMF.
Code lines: 9
Contained by: file `stellar_populations.initial_mass_functions.BPASS.F90`

interface: `initialmassfunctionbpass`
Description: Constructors for the `bpass` initial mass function class.
Code lines: 4
Contained by: file `stellar_populations.initial_mass_functions.BPASS.F90`

file: `stellar_populations.initial_mass_functions.Baugh2005TopHeavy.F90`
Description: Implements a stellar initial mass function class for the top-heavy stellar initial mass function from Baugh et al. [2005].
Code lines: 70

function: `baugh2005topheavyconstructorinternal`
Description: Internal constructor for the `baugh2005TopHeavy` initial mass function.
Code lines: 7
Contained by: file `stellar_populations.initial_mass_functions.Baugh2005TopHeavy.F90`

function: `baugh2005topheavyconstructorparameters`
Description: Constructor for the `baugh2005TopHeavy` initial mass function class which takes a parameter list as input.
Code lines: 10
Contained by: file `stellar_populations.initial_mass_functions.Baugh2005TopHeavy.F90`
Modules used: `input_parameters`

function: `baugh2005topheavylabel`
Description: Return a label for this IMF.
Code lines: 9
Contained by: file `stellar_populations.initial_mass_functions.Baugh2005TopHeavy.F90`

interface: `initialmassfunctionbaugh2005topheavy`
Description: Constructors for the `baugh2005TopHeavy` initial mass function class.
Code lines: 4
Contained by: file `stellar_populations.initial_mass_functions.Baugh2005TopHeavy.F90`

file: `stellar_populations.initial_mass_functions.Chabrier2001.F90`
Description: Implements a stellar initial mass function class based on Chabrier [2001].
Code lines: 186

function: `chabrier2001constructorinternal`
Description: Internal constructor for the `chabrier2001` initial mass function.
Code lines: 16

Contained by: file stellar_populations.initial_mass_functions.Chabrier2001.F90
Modules used: error_functions numerical_constants_math

function: chabrier2001constructorparameters
Description: Constructor for the chabrier2001 initial mass function class which takes a parameter list as input.
Code lines: 59
Contained by: file stellar_populations.initial_mass_functions.Chabrier2001.F90
Modules used: input_parameters

function: chabrier2001label
Description: Return a label for this IMF.
Code lines: 9
Contained by: file stellar_populations.initial_mass_functions.Chabrier2001.F90

function: chabrier2001massmaximum
Description: Return the maximum mass of stars in the Chabrier [2001] IMF.
Code lines: 7
Contained by: file stellar_populations.initial_mass_functions.Chabrier2001.F90

function: chabrier2001massminimum
Description: Return the minimum mass of stars in the Chabrier [2001] IMF.
Code lines: 7
Contained by: file stellar_populations.initial_mass_functions.Chabrier2001.F90

function: chabrier2001phi
Description: Evaluate the Chabrier [2001] stellar initial mass function.
Code lines: 14
Contained by: file stellar_populations.initial_mass_functions.Chabrier2001.F90

subroutine: chabrier2001tabulate
Description: Construct and return a tabulation of the Chabrier [2001] IMF.
Code lines: 17
Contained by: file stellar_populations.initial_mass_functions.Chabrier2001.F90

interface: initialmassfunctionchabrier2001
Description: Constructors for the chabrier2001 initial mass function class.
Code lines: 4
Contained by: file stellar_populations.initial_mass_functions.Chabrier2001.F90

file: stellar_populations.initial_mass_functions.F90
Description: Contains a module which implements a class for stellar initial mass functions.
Code lines: 62

module: stellar_populations_initial_mass_functions
Description: Implements a class for stellar initial mass functions.
Code lines: 40
Contained by: file stellar_populations.initial_mass_functions.F90
Modules used: tables
Used by: program benchmark_stellar_populations_luminosities subroutine galacticus_function_classes_destroy

subroutine galacticus_state_retrieve
file stellar_-populations.spectra.FSPS.F90
program test_initial_mass_functions
program test_stellar_populations_-luminosities
subroutine galacticus_state_store
file stellar_populations.standard.F90
program test_stellar_populations

file: stellar_populations.initial_mass_functions.Kennicutt1983.F90
Description: Implements a stellar initial mass function class for the Kennicutt [1983] IMF.
Code lines: 70

interface: initialmassfunctionkennicutt1983
Description: Constructors for the kennicutt1983 initial mass function class.
Code lines: 4
Contained by: file stellar_populations.initial_mass_functions.Kennicutt1983.F90

function: kennicutt1983constructorinternal
Description: Internal constructor for the kennicutt1983 initial mass function.
Code lines: 7
Contained by: file stellar_populations.initial_mass_functions.Kennicutt1983.F90

function: kennicutt1983constructorparameters
Description: Constructor for the kennicutt1983 initial mass function class which takes a parameter list as input.
Code lines: 10
Contained by: file stellar_populations.initial_mass_functions.Kennicutt1983.F90
Modules used: input_parameters

function: kennicutt1983label
Description: Return a label for this IMF.
Code lines: 9
Contained by: file stellar_populations.initial_mass_functions.Kennicutt1983.F90

file: stellar_populations.initial_mass_functions.Kroupa2001.F90
Description: Implements a stellar initial mass function class for the Kroupa [2001] IMF.
Code lines: 70

interface: initialmassfunctionkroupa2001
Description: Constructors for the kroupa2001 initial mass function class.
Code lines: 4
Contained by: file stellar_populations.initial_mass_functions.Kroupa2001.F90

function: kroupa2001constructorinternal
Description: Internal constructor for the kroupa2001 initial mass function.
Code lines: 7
Contained by: file stellar_populations.initial_mass_functions.Kroupa2001.F90

function: kroupa2001constructorparameters
Description: Constructor for the kroupa2001 initial mass function class which takes a parameter list as input.

Code lines: 10
Contained by: file stellar_populations.initial_mass_functions.Kroupa2001.F90
Modules used: input_parameters

function: kroupa2001label
Description: Return a label for this IMF.
Code lines: 9
Contained by: file stellar_populations.initial_mass_functions.Kroupa2001.F90

file: stellar_populations.initial_mass_functions.MillerScalo1979.F90
Description: Implements a stellar initial mass function class for the Miller and Scalo [1979] IMF.
Code lines: 70

interface: initialmassfunctionmillerscalo1979
Description: Constructors for the millerScalo1979 initial mass function class.
Code lines: 4
Contained by: file stellar_populations.initial_mass_functions.MillerScalo1979.F90

function: millerscalo1979constructorinternal
Description: Internal constructor for the millerScalo1979 initial mass function.
Code lines: 7
Contained by: file stellar_populations.initial_mass_functions.MillerScalo1979.F90

function: millerscalo1979constructorparameters
Description: Constructor for the millerScalo1979 initial mass function class which takes a parameter list as input.
Code lines: 10
Contained by: file stellar_populations.initial_mass_functions.MillerScalo1979.F90
Modules used: input_parameters

function: millerscalo1979label
Description: Return a label for this IMF.
Code lines: 9
Contained by: file stellar_populations.initial_mass_functions.MillerScalo1979.F90

file: stellar_populations.initial_mass_functions.Salpeter1955.F90
Description: Implements a stellar initial mass function class for the Salpeter [1955] IMF.
Code lines: 70

interface: initialmassfunctionsalpeter1955
Description: Constructors for the salpeter1955 initial mass function class.
Code lines: 4
Contained by: file stellar_populations.initial_mass_functions.Salpeter1955.F90

function: salpeter1955constructorinternal
Description: Internal constructor for the salpeter1955 initial mass function.
Code lines: 7
Contained by: file stellar_populations.initial_mass_functions.Salpeter1955.F90

function: `salpeter1955constructorparameters`
Description: Constructor for the `salpeter1955` initial mass function class which takes a parameter list as input.
Code lines: 10
Contained by: file `stellar_populations.initial_mass_functions.Salpeter1955.F90`
Modules used: `input_parameters`

function: `salpeter1955label`
Description: Return a label for this IMF.
Code lines: 9
Contained by: file `stellar_populations.initial_mass_functions.Salpeter1955.F90`

file: `stellar_populations.initial_mass_functions.Scalo1986.F90`
Description: Implements a stellar initial mass function class for the Scalo [1986] IMF.
Code lines: 70

interface: `initialmassfunctionscalo1986`
Description: Constructors for the `scalo1986` initial mass function class.
Code lines: 4
Contained by: file `stellar_populations.initial_mass_functions.Scalo1986.F90`

function: `scalo1986constructorinternal`
Description: Internal constructor for the `scalo1986` initial mass function.
Code lines: 7
Contained by: file `stellar_populations.initial_mass_functions.Scalo1986.F90`

function: `scalo1986constructorparameters`
Description: Constructor for the `scalo1986` initial mass function class which takes a parameter list as input.
Code lines: 10
Contained by: file `stellar_populations.initial_mass_functions.Scalo1986.F90`
Modules used: `input_parameters`

function: `scalo1986label`
Description: Return a label for this IMF.
Code lines: 9
Contained by: file `stellar_populations.initial_mass_functions.Scalo1986.F90`

file: `stellar_populations.initial_mass_functions.piecewise_power_law.F90`
Description: Implements a stellar initial mass function class for piecewise power-law IMFs.
Code lines: 191

interface: `initialmassfunctionpiecewisepowerlaw`
Description: Constructors for the `piecewisePowerLaw` initial mass function class.
Code lines: 4
Contained by: file `stellar_populations.initial_mass_functions.piecewise_power_law.F90`

function: `piecewisepowerlawconstructorinternal`

Description: Internal constructor for the `piecewisePowerLaw` initial mass function.
Code lines: 46
Contained by: file stellar_populations.initial_mass_functions.piecewise_power_law.F90
Modules used: array_utilities galacticus_error

function: `piecewisepowerlawconstructorparameters`
Description: Constructor for the `piecewisePowerLaw` initial mass function class which takes a parameter list as input.
Code lines: 32
Contained by: file stellar_populations.initial_mass_functions.piecewise_power_law.F90
Modules used: input_parameters

function: `piecewisepowerlawlabel`
Description: Return a label for this IMF.
Code lines: 9
Contained by: file stellar_populations.initial_mass_functions.piecewise_power_law.F90

function: `piecewisepowerlawmassmaximum`
Description: Return the maximum mass of stars in a piecewise power-law IMF.
Code lines: 7
Contained by: file stellar_populations.initial_mass_functions.piecewise_power_law.F90

function: `piecewisepowerlawmassminimum`
Description: Return the minimum mass of stars in the Chabrier [2001] IMF.
Code lines: 7
Contained by: file stellar_populations.initial_mass_functions.piecewise_power_law.F90

function: `piecewisepowerlawphi`
Description: Evaluate a piecewise power-law stellar initial mass function.
Code lines: 15
Contained by: file stellar_populations.initial_mass_functions.piecewise_power_law.F90

subroutine: `piecewisepowerlawtabulate`
Description: Construct and return a tabulation of a piecewise power-law IMF.
Code lines: 17
Contained by: file stellar_populations.initial_mass_functions.piecewise_power_law.F90

file: `stellar_populations.luminosities.F90`
Description: Contains a module which implements calculations of stellar population luminosities in the AB magnitude system.
Code lines: 551

module: `stellar_population_luminosities`
Description: Implements calculations of stellar population luminosities in the AB magnitude system.
Code lines: 529
Contained by: file stellar_populations.luminosities.F90
Modules used: abundances_structure fgsl iso_c_binding iso_varying_string locks stellar_population_spectra_postprocess

Used by: program benchmark_stellar_populations_luminosities; function luminosityintegrand; program test_stellar_populations_luminosities; function sedfitevaluate; subroutine stellar_luminosities_set

type: `luminositytable`

Description: Structure for holding tables of simple stellar population luminosities.
Code lines: 10
Contained by: module stellar_population_luminosities

function: `stellar_population_luminosity`

Description: Returns the luminosity for a $1M_\odot$ simple `stellarPopulation_` of given `abundances` and `age` and observed through the filter specified by `filterIndex`.
Code lines: 72
Contained by: module stellar_population_luminosities
Modules used: galacticus_error; iso_c_binding; numerical_interpolation; stellar_populations

subroutine: `stellar_population_luminosity_tabulate`

Description: Tabulate stellar population luminosity in the given filters.
Code lines: 342
Contained by: module stellar_population_luminosities
Modules used: file_utilities; galacticus_display; galacticus_error; galacticus_paths; input_parameters; instruments_filters; io_hdf5; iso_c_binding; memory_management; numerical_constants_astronomical; numerical_integration; stellar_population_spectra; stellar_populations; string_handling

function: `filter_luminosity_integrand`

Description: Integrand for the luminosity through a given filter.
Code lines: 17
Contained by: subroutine stellar_population_luminosity_tabulate
Modules used: instruments_filters

function: `filter_luminosity_integrand_ab`

Description: Integrand for the luminosity of a zeroth magnitude (AB) source through a given filter.
Code lines: 11
Contained by: subroutine stellar_population_luminosity_tabulate
Modules used: instruments_filters; numerical_constants_astronomical

subroutine: `stellar_population_luminosity_track`

Description: Returns the luminosity for a $1M_\odot$ simple stellar population of given `abundances` drawn from the given `stellarPopulation` and observed through the filter specified by `filterIndex`, for all available ages.
Code lines: 55
Contained by: module stellar_population_luminosities
Modules used: galacticus_error; iso_c_binding

memory_management numerical_interpolation
stellar_populations

file: `stellar_populations.properties.F90`
Description: Contains a module which implements a class for computing properties of stellar populations.
Code lines: 71

module: `stellar_population_properties`
Description: Implements a class for computing properties of stellar populations.
Code lines: 49
Contained by: file stellar_populations.properties.F90
Modules used: abundances_structure galacticus_nodes histories
Used by: subroutine galacticus_function_classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; module node_component_disk_standard; module node_component_disk_very_simple; module node_component_spheroid_standard; module node_component_spheroid_very_simple

file: `stellar_populations.properties.instantaneous.F90`
Description: Implements a stellar population properties class based on the instantaneous recycling approximation.
Code lines: 200
Modules used: iso_c_binding stellar_luminosities_structure stellar_population_selectors

function: `instantaneousconstructorinternal`
Description: Internal constructor for the `instantaneous` stellar population properties class.
Code lines: 16
Contained by: file stellar_populations.properties.instantaneous.F90
Modules used: atomic_data

function: `instantaneousconstructorparameters`
Description: Constructor for the `instantaneous` stellar population properties class which takes a parameter list as input.
Code lines: 14
Contained by: file stellar_populations.properties.instantaneous.F90
Modules used: input_parameters

subroutine: `instantaneousdestructor`
Description: Destructor for the `instantaneous` stellar population properties class.
Code lines: 7
Contained by: file stellar_populations.properties.instantaneous.F90

function: `instantaneoushistorycount`
Description: Returns the number of histories required by the instantaneous stellar populations properties class.
Code lines: 8

Contained by: file stellar_populations.properties.instantaneous.F90

subroutine: instantaneoushistorycreate

Description: Create any history required for storing stellar population properties. The instantaneous method requires none, so don't create one.
Code lines: 10
Contained by: file stellar_populations.properties.instantaneous.F90

subroutine: instantaneousrates

Description: Return stellar population property rates of change given a star formation rate and fuel abundances.
Code lines: 69
Contained by: file stellar_populations.properties.instantaneous.F90
Modules used: galactic_structure_options galacticus_nodes stellar_feedback stellar_populations

subroutine: instantaneousscales

Description: Set the scalings for error control on the absolute values of stellar population properties. The instantaneous method requires none, so just return.
Code lines: 11
Contained by: file stellar_populations.properties.instantaneous.F90

interface: stellarpopulationpropertiesinstantaneous

Description: Constructors for the instantaneous stellar population class.
Code lines: 4
Contained by: file stellar_populations.properties.instantaneous.F90

file: stellar_populations.properties.noninstantaneous.F90

Description: Implements a stellar population properties class based on the noninstantaneous recycling approximation.
Code lines: 262
Modules used: output_times stellar_population_selectors

function: noninstantaneousconstructorinternal

Description: Internal constructor for the noninstantaneous stellar population properties class.
Code lines: 22
Contained by: file stellar_populations.properties.noninstantaneous.F90

function: noninstantaneousconstructorparameters

Description: Constructor for the noninstantaneous stellar population properties class which takes a parameter list as input.
Code lines: 26
Contained by: file stellar_populations.properties.noninstantaneous.F90
Modules used: input_parameters

subroutine: noninstantaneousdestructor

Description: Destructor for the noninstantaneous stellar population properties class.
Code lines: 8
Contained by: file stellar_populations.properties.noninstantaneous.F90

function: `noninstantaneoushistorycount`

Description: Returns the number of histories required by the noninstantaneous stellar populations properties class.
Code lines: 7
Contained by: file stellar_populations.properties.noninstantaneous.F90

subroutine: `noninstantaneoushistorycreate`

Description: Create any history required for storing stellar population properties.
Code lines: 18
Contained by: file stellar_populations.properties.noninstantaneous.F90
Modules used: galacticus_nodes, numerical_ranges

subroutine: `noninstantaneousrates`

Description: Return an array of stellar population property rates of change given a star formation rate and fuel abundances.
Code lines: 78
Contained by: file stellar_populations.properties.noninstantaneous.F90
Modules used: fgsl, galacticus_nodes, iso_c_binding, numerical_interpolation, stellar_luminosities_structure, stellar_populations

subroutine: `noninstantaneousscales`

Description: Set the scalings for error control on the absolute values of stellar population properties.
Code lines: 34
Contained by: file stellar_populations.properties.noninstantaneous.F90
Modules used: memory_management, stellar_feedback

interface: `stellarpopulationpropertiesnoninstantaneous`

Description: Constructors for the `noninstantaneous` stellar population class.
Code lines: 4
Contained by: file stellar_populations.properties.noninstantaneous.F90

file: `stellar_populations.selectors.F90`

Description: Contains a module which implements a class for selecting stellar populations.
Code lines: 50

module: `stellar_population_selectors`

Description: Implements a class for selecting stellar populations.
Code lines: 28
Contained by: file stellar_populations.selectors.F90
Modules used: abundances_structure, galacticus_nodes, stellar_populations
Used by: subroutine galacticus_function_classes_destroy, subroutine galacticus_state_retrieve, subroutine galacticus_state_store, file models.likelihoods.SED_fit.F90, file radiation.intergalactic_background.internal.F90, file stellar_populations.properties.instantaneous.F90

file stellar_-
populations.properties.noninstantaneous.F90

file: stellar_populations.selectors.disk_spheroid.F90
Description: Implements a stellar population selector class which returns a different population for disks and spheroids.
Code lines: 113

function: diskspheroidconstructorinternal
Description: Internal constructor for the diskSpheroid stellar population selector class.
Code lines: 8
Contained by: file stellar_populations.selectors.disk_spheroid.F90

function: diskspheroidconstructorparameters
Description: Constructor for the diskSpheroid stellar population class which takes a parameter list as input.
Code lines: 16
Contained by: file stellar_populations.selectors.disk_spheroid.F90
Modules used: input_parameters stellar_populations

subroutine: diskspheroiddestructor
Description: Destructor for the diskSpheroid stellar population selector class.
Code lines: 8
Contained by: file stellar_populations.selectors.disk_spheroid.F90

function: diskspheroidisstarformationratedependent
Description: Return false indicating that stellar population selection is not dependent on star formation rate.
Code lines: 8
Contained by: file stellar_populations.selectors.disk_spheroid.F90

function: diskspheroidselect
Description: Return a diskSpheroid stellar population.
Code lines: 21
Contained by: file stellar_populations.selectors.disk_spheroid.F90
Modules used: galacticus_error galacticus_nodes

interface: stellarpopulationselectordiskspheroid
Description: Constructors for the diskSpheroid stellar population selector class.
Code lines: 4
Contained by: file stellar_populations.selectors.disk_spheroid.F90

file: stellar_populations.selectors.fixed.F90
Description: Implements a stellar population selector class which returns a fixed population.
Code lines: 101

function: fixedconstructorinternal
Description: Internal constructor for the fixed stellar population selector class.
Code lines: 8

Contained by: file stellar_populations.selectors.fixed.F90

function: fixedconstructorparameters
Description: Constructor for the fixed stellar population class which takes a parameter list as input.
Code lines: 14
Contained by: file stellar_populations.selectors.fixed.F90
Modules used: input_parameters stellar_populations

subroutine: fixeddestructor
Description: Destructor for the fixed stellar population selector class.
Code lines: 7
Contained by: file stellar_populations.selectors.fixed.F90

function: fixedisstarformationratedependent
Description: Return false indicating that stellar population selection is not dependent on star formation rate.
Code lines: 8
Contained by: file stellar_populations.selectors.fixed.F90

function: fixedselect
Description: Return a fixed stellar population.
Code lines: 12
Contained by: file stellar_populations.selectors.fixed.F90

interface: stellarpopulationselectorfixed
Description: Constructors for the fixed stellar population selector class.
Code lines: 4
Contained by: file stellar_populations.selectors.fixed.F90

file: stellar_populations.spectra.F90
Description: Contains a module that provides a class implementing stellar population spectra.
Code lines: 64

module: stellar_population_spectra
Description: Provides a class implementing stellar population spectra.
Code lines: 40
Contained by: file stellar_populations.spectra.F90
Modules used: abundances_structure
Used by:
program benchmark_stellar_populations_luminosities
subroutine galacticus_function_classes_destroy
subroutine galacticus_state_retrieve
subroutine galacticus_state_store
subroutine stellar_luminosities_sed_top_hat_step
subroutine stellar_luminosities_special_cases
function intergalacticbackgroundinternalupdate
module stellar_populations
subroutine stellar_population_luminosity_tabulate
file stellar_populations.standard.F90
program test_stellar_populations
program test_stellar_populations_luminosities

file: `stellar_populations.spectra.FSPS.F90`
Description: Implements a stellar population spectra class which utilizes the FSPS package [Conroy et al., 2009].
Code lines: 128
Modules used: `stellar_populations_initial_mass_functions`

function: `fspsconstructorinternal`
Description: Internal constructor for the FSPS stellar spectra class.
Code lines: 11
Contained by: file `stellar_populations.spectra.FSPS.F90`
Modules used: `galacticus_paths`

function: `fspsconstructorparameters`
Description: Constructor for the FSPS stellar spectra class which takes a parameter set as input.
Code lines: 21
Contained by: file `stellar_populations.spectra.FSPS.F90`

subroutine: `fspsdestructor`
Description: Destructor for the `FSPS` stellar population class.
Code lines: 7
Contained by: file `stellar_populations.spectra.FSPS.F90`

subroutine: `fspsreadfile`
Description: Ensure that the requested stellar population has been generated.
Code lines: 39
Contained by: file `stellar_populations.spectra.FSPS.F90`
Modules used: `file_utilities` `interfaces_fsps` `io_hdf5` `tables`

interface: `stellarpopulationspectrafsps`
Description: Constructors for the FSPS stellar spectra class.
Code lines: 4
Contained by: file `stellar_populations.spectra.FSPS.F90`

file: `stellar_populations.spectra.dust_attenuation.Calzetti2000.F90`
Description: Implements calculations of attenuation of stellar spectra using the model of Calzetti et al. [2000].
Code lines: 74

function: `calzetti2000attenuation`
Description: Return attenuation of stellar spectra according to the model of Calzetti et al. [2000].
Code lines: 22
Contained by: file `stellar_populations.spectra.dust_attenuation.Calzetti2000.F90`
Modules used: `numerical_constants_units`

function: `calzetti2000constructorparameters`
Description: Constructor for the `calzetti2000` stellar spectra dust attenuation class which takes a parameter set as input.

Code lines: 11
Contained by: file stellar_populations.spectra.dust_attenuation.Calzetti2000.F90
Modules used: input_parameters

interface: stellarspectradustattenuationcalzetti2000
Description: Constructors for the "calzetti2000" stellar spectra dust attenuation class.
Code lines: 3
Contained by: file stellar_populations.spectra.dust_attenuation.Calzetti2000.F90

file: stellar_populations.spectra.dust_attenuation.Cardelli1989.F90
Description: Implements calculations of attenuation of stellar spectra using the model of Cardelli et al. [1989].
Code lines: 152

function: cardelli1989a
Description: Return fitting function $a(x)$ for the dust attenuation model of Cardelli et al. [1989].
Code lines: 23
Contained by: file stellar_populations.spectra.dust_attenuation.Cardelli1989.F90

function: cardelli1989attenuation
Description: Return attenuation of stellar spectra according to the model of Cardelli et al. [1989].
Code lines: 12
Contained by: file stellar_populations.spectra.dust_attenuation.Cardelli1989.F90
Modules used: numerical_constants_units

function: cardelli1989b
Description: Return fitting function $a(x)$ for the dust attenuation model of Cardelli et al. [1989].
Code lines: 23
Contained by: file stellar_populations.spectra.dust_attenuation.Cardelli1989.F90

function: cardelli1989constructorinternal
Description: Constructor for the "cardelli1989" stellar spectra dust attenuation class.
Code lines: 8
Contained by: file stellar_populations.spectra.dust_attenuation.Cardelli1989.F90

function: cardelli1989constructorparameters
Description: Constructor for the cardelli1989 stellar spectra dust attenuation class which takes a parameter set as input.
Code lines: 20
Contained by: file stellar_populations.spectra.dust_attenuation.Cardelli1989.F90
Modules used: input_parameters

interface: stellarspectradustattenuationcardelli1989
Description: Constructors for the "cardelli1989" stellar spectra dust attenuation class.
Code lines: 4
Contained by: file stellar_populations.spectra.dust_attenuation.Cardelli1989.F90

file: stellar_populations.spectra.dust_attenuation.CharlotFall2000.F90
Description: Implements calculations of attenuation of stellar spectra using the model of Charlot and Fall [2000].
Code lines: 123

function: `charlotfall2000attenuation`
Description: Return attenuation of stellar spectra according to the model of Charlot and Fall [2000]. Note that the V-band attenuation is taken to be that due to the ISM alone (i.e. not including birthclouds).
Code lines: 17
Contained by: file `stellar_populations.spectra.dust_attenuation.CharlotFall2000.F90`

function: `charlotfall2000constructorinternal`
Description: Constructor for the `charlotFall2000` stellar spectra dust attenuation class.
Code lines: 8
Contained by: file `stellar_populations.spectra.dust_attenuation.CharlotFall2000.F90`

function: `charlotfall2000constructorparameters`
Description: Default constructor for the `charlotFall2000` stellar spectra dust attenuatio class.
Code lines: 42
Contained by: file `stellar_populations.spectra.dust_attenuation.CharlotFall2000.F90`

function: `charlotfall2000isagedependent`
Description: Return true since attenuation is age-dependent in the Charlot and Fall [2000] dust attenuation model.
Code lines: 8
Contained by: file `stellar_populations.spectra.dust_attenuation.CharlotFall2000.F90`

interface: `stellarspectradustattenuationcharlotfall2000`
Description: Constructors for the `charlotFall2000` stellar spectra dust attenuation class.
Code lines: 4
Contained by: file `stellar_populations.spectra.dust_attenuation.CharlotFall2000.F90`

file: `stellar_populations.spectra.dust_attenuation.F90`
Description: Contains a module that provides a class implementing dust attenuation of stellar spectra.
Code lines: 57

module: `stellar_spectra_dust_attenuations`
Description: Provides a class implementing dust attenuation of stellar spectra.
Code lines: 35
Contained by: file `stellar_populations.spectra.dust_attenuation.F90`
Used by:

subroutine `galacticus_function_classes_destroy`	file `galacticus.output.analyses.luminosity_function.Halpha.F90`
subroutine `galacticus_state_retrieve`	subroutine `galacticus_state_store`
function `sedfitevaluate`	file `nodes.property_extractor.luminosity_emission_line.F90`

file: `stellar_populations.spectra.dust_attenuation.Gordon2003.F90`
Description: Implements calculations of attenuation of stellar spectra using the model of Gordon et al. [2003].
Code lines: 90

function: `gordon2003constructorinternal`
Description: Constructor for the "gordon2003" stellar spectra dust attenuation class.
Code lines: 19
Contained by: file `stellar_populations.spectra.dust_attenuation.Gordon2003.F90`
Modules used: `galacticus_error` `table_labels`

function: `gordon2003constructorparameters`
Description: Default constructor for the "gordon2003" stellar spectra dust attenuation class.
Code lines: 19
Contained by: file `stellar_populations.spectra.dust_attenuation.Gordon2003.F90`
Modules used: `input_parameters`

interface: `stellarspectradustattenuationgordon2003`
Description: Constructors for the "gordon2003" stellar spectra dust attenuation class.
Code lines: 4
Contained by: file `stellar_populations.spectra.dust_attenuation.Gordon2003.F90`

file: `stellar_populations.spectra.dust_attenuation.Prevot-Bouchet.F90`
Description: Implements calculations of attenuation of stellar spectra using the model of Prevot et al. [1984] and Bouchet et al. [1985].
Code lines: 74

function: `prevotbouchetconstructorinternal`
Description: Constructor for the "prevotBouchet" stellar spectra dust attenuation class. Data read directly from Table 3 of Bouchet et al. [1985].
Code lines: 12
Contained by: file `stellar_populations.spectra.dust_attenuation.Prevot-Bouchet.F90`
Modules used: `galacticus_error` `table_labels`

function: `prevotbouchetconstructorparameters`
Description: Constructor for the "prevotBouchet" stellar spectra dust attenuation class which takes a parameter set as input.
Code lines: 21
Contained by: file `stellar_populations.spectra.dust_attenuation.Prevot-Bouchet.F90`
Modules used: `galacticus_error` `input_parameters` `table_labels`

interface: `stellarspectradustattenuationprevotbouchet`
Description: Constructors for the "prevotBouchet" stellar spectra dust attenuation class.
Code lines: 4
Contained by: file `stellar_populations.spectra.dust_attenuation.Prevot-Bouchet.F90`

file: `stellar_populations.spectra.dust_attenuation.WittGordon2000.F90`
Description: Implements calculations of attenuation of stellar spectra using the model of Witt and Gordon [2000].
Code lines: 91

interface: `stellarspectradustattenuationwittgordon2000`

Description: Constructors for the “wittGordon2003” stellar spectra dust attenuation class.
Code lines: 4
Contained by: file stellar_populations.spectra.dust_attenuation.WittGordon2000.F90

function: wittgordon2003constructorinternal
Description: Constructor for the “wittGordon2003” stellar spectra dust attenuation class.
Code lines: 21
Contained by: file stellar_populations.spectra.dust_attenuation.WittGordon2000.F90
Modules used: array_utilities galacticus_error numerical_constants_astronomical numerical_constants_units table_labels

function: wittgordon2003constructorparameters
Description: Default constructor for the “wittGordon2003” stellar spectra dust attenuation class.
Code lines: 19
Contained by: file stellar_populations.spectra.dust_attenuation.WittGordon2000.F90
Modules used: input_parameters

file: stellar_populations.spectra.dust_attenuation.null.F90
Description: Implements calculations of zero dust attenuation of stellar spectra.
Code lines: 60

interface: stellarspectradustattenuationzero
Description: Constructors for the “zero” stellar spectra dust attenuation class.
Code lines: 3
Contained by: file stellar_populations.spectra.dust_attenuation.null.F90

function: zeroattenuation
Description: Return a zero attenuation.
Code lines: 9
Contained by: file stellar_populations.spectra.dust_attenuation.null.F90

function: zeroconstructorparameters
Description: Default constructor for the “zero” stellar spectra dust attenuation class.
Code lines: 10
Contained by: file stellar_populations.spectra.dust_attenuation.null.F90
Modules used: input_parameters

file: stellar_populations.spectra.dust_attenuation.tabulated.F90
Description: Implements calculations of attenuation of stellar spectra using a tabulation.
Code lines: 62
Modules used: tables

type: stellarspectradustattenuationtabulated
Description: A class implementing calculations of attenuation of stellar spectra using a tabulated relation.
Code lines: 7
Contained by: file stellar_populations.spectra.dust_attenuation.tabulated.F90

function: tabulatedattenuation

Description: Return attenuation of stellar spectra according to the model of Gordon et al. [2003].
Code lines: 13
Contained by: file stellar_populations.spectra.dust_attenuation.tabulated.F90
Modules used: numerical_constants_units

subroutine: tabulateddestructor
Description: Destructor for the "tabulated" stellar spectra dust attenuation class.
Code lines: 8
Contained by: file stellar_populations.spectra.dust_attenuation.tabulated.F90
Modules used: input_parameters

file: stellar_populations.spectra.file.F90
Description: Implements a file-based stellar population spectra class.
Code lines: 370
Modules used: fgsl

function: fileconstructorinternal
Description: Internal constructor for the file stellar spectra class.
Code lines: 11
Contained by: file stellar_populations.spectra.file.F90
Modules used: galacticus_error

function: fileconstructorparameters
Description: Constructor for the file stellar spectra class which takes a parameter set as input.
Code lines: 27
Contained by: file stellar_populations.spectra.file.F90
Modules used: galacticus_paths

function: fileluminosity
Description: Return the luminosity (in units of $L_\odot$ Hz^{-1}) for a stellar population with composition abundances, of the given age (in Gyr) and the specified wavelength (in Angstroms). This is found by interpolating in tabulated spectra.
Code lines: 82
Contained by: file stellar_populations.spectra.file.F90
Modules used: abundances_structure, galacticus_error, iso_c_binding, numerical_interpolation

subroutine: filereadfile
Description: Read a file of simple stellar population spectra.
Code lines: 40
Contained by: file stellar_populations.spectra.file.F90
Modules used: file_utilities, galacticus_error, io_hdf5, memory_management

subroutine: filetabulation
Description: Return a tabulation of ages and metallicities at which stellar spectra should be tabulated.
Code lines: 19
Contained by: file stellar_populations.spectra.file.F90
Modules used: memory_management, numerical_constants_astronomical

function: `filewavelengthinterval`
Description: Return a tabulation of wavelengths at which stellar spectra should be tabulated.
Code lines: 28
Contained by: file stellar_populations.spectra.file.F90
Modules used: memory_management numerical_constants_astronomical

subroutine: `filewavelengths`
Description: Return a tabulation of wavelengths at which stellar spectra should be tabulated.
Code lines: 16
Contained by: file stellar_populations.spectra.file.F90
Modules used: memory_management numerical_constants_astronomical

type: `spectraltable`
Description: Structure to hold tabulated stellar population data.
Code lines: 11
Contained by: file stellar_populations.spectra.file.F90

subroutine: `spectraltabledestructor1d`
Description: Destructor for the stellar spectra table class.
Code lines: 11
Contained by: file stellar_populations.spectra.file.F90
Modules used: numerical_interpolation

subroutine: `spectraltabledestructorscalar`
Description: Destructor for the stellar spectra table class.
Code lines: 11
Contained by: file stellar_populations.spectra.file.F90
Modules used: numerical_interpolation

interface: `stellarpopulationspectrafile`
Description: Constructors for the file stellar spectra class.
Code lines: 4
Contained by: file stellar_populations.spectra.file.F90

file: `stellar_populations.spectra.postprocess.F90`
Description: Contains a module which implements a class for stellar spectra postprocessors.
Code lines: 71

module: `stellar_population_spectra_postprocess`
Description: Implements a class for stellar spectra postprocessors.
Code lines: 49
Contained by: file stellar_populations.spectra.postprocess.F90
Used by: program benchmark_stellar_populations_luminosities; subroutine galacticus_function_classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; file models.likelihoods.SED_fit.F90; module stellar_luminosities_structure; subroutine stellar_luminosities_state_restore; module stellar_population_luminosities

program test_inoue2014

program test_stellar_populations_-luminosities

type: stellarpopulationspectrapostprocessorlist
Description: Type used to build linked list of stellar population spectra postprocessors.
Code lines: 5
Contained by: module stellar_population_spectra_postprocess

subroutine: stellarpopulationspectrapostprocessorlistdestructor
Description: Destructor for elements of stellar population spectra postprocessor lists.
Code lines: 7
Contained by: module stellar_population_spectra_postprocess

file: stellar_populations.spectra.postprocess.Inoue2014.F90
Description: An implementation of a spectrum postprocessor that applies the Inoue et al. [2014] calculation of the attenuation of spectra by the intergalactic medium.
Code lines: 131

function: inoue2014constructorparameters
Description: Constructor for the inoue2014 stellar population spectra postprocessor class which takes a parameter list as input.
Code lines: 11
Contained by: file stellar_populations.spectra.postprocess.Inoue2014.F90
Modules used: input_parameters

function: inoue2014multiplier
Description: Apply the Inoue et al. [2014] calculation of the attenuation of spectra by the intergalactic medium.
Code lines: 74
Contained by: file stellar_populations.spectra.postprocess.Inoue2014.F90
Modules used: factorials gamma_functions numerical_constants_atomic

interface: stellarpopulationspectrapostprocessorinoue2014
Description: Constructors for the inoue2014 stellar population spectra postprocessor class.
Code lines: 3
Contained by: file stellar_populations.spectra.postprocess.Inoue2014.F90

file: stellar_populations.spectra.postprocess.Lyman_continuum_suppress.F90
Description: An implementation of a spectrum postprocessor that suppresses the Lyman continuum.
Code lines: 66

function: lycsuppressconstructorparameters
Description: Constructor for the lycSuppress stellar population spectra postprocessor class which takes a parameter list as input.
Code lines: 11
Contained by: file stellar_populations.spectra.postprocess.Lyman_continuum_suppress.F90
Modules used: input_parameters

function: lycsuppressmultiplier

Description: Suppress the Lyman continuum in a spectrum.
Code lines: 14
Contained by: file stellar_populations.spectra.postprocess.Lyman_continuum_suppress.F90
Modules used: numerical_constants_atomic

interface: stellarpopulationspectrapostprocessorlycsuppress
Description: Constructors for the lycSuppress stellar population spectra postprocessor class.
Code lines: 3
Contained by: file stellar_populations.spectra.postprocess.Lyman_continuum_suppress.F90

file: stellar_populations.spectra.postprocess.Madau1995.F90
Description: An implementation of a spectrum postprocessor that applies the Madau [1995] calculation of the attenuation of spectra by the intergalactic medium.
Code lines: 92

function: madau1995constructorparameters
Description: Constructor for the madau1995 stellar population spectra postprocessor class which takes a parameter list as input.
Code lines: 11
Contained by: file stellar_populations.spectra.postprocess.Madau1995.F90
Modules used: input_parameters

function: madau1995multiplier
Description: Suppress the Lyman continuum in a spectrum.
Code lines: 39
Contained by: file stellar_populations.spectra.postprocess.Madau1995.F90
Modules used: numerical_constants_atomic

interface: stellarpopulationspectrapostprocessormadau1995
Description: Constructors for the madau1995 stellar population spectra postprocessor class.
Code lines: 3
Contained by: file stellar_populations.spectra.postprocess.Madau1995.F90

file: stellar_populations.spectra.postprocess.Meiksin2006.F90
Description: An implementation of a spectrum postprocessor that applies the Meiksin [2006] calculation of the attenuation of spectra by the intergalactic medium.
Code lines: 139

function: meiksin2006constructorparameters
Description: Constructor for the meiksin2006 stellar population spectra postprocessor class which takes a parameter list as input.
Code lines: 11
Contained by: file stellar_populations.spectra.postprocess.Meiksin2006.F90
Modules used: input_parameters

function: meiksin2006multiplier
Description: Suppress the Lyman continuum in a spectrum.
Code lines: 86
Contained by: file stellar_populations.spectra.postprocess.Meiksin2006.F90

Modules used: factorials gamma_functions numerical_constants_atomic

interface: stellarpopulationspectrapostprocessormeiksin2006
Description: Constructors for the meiksin2006 stellar population spectra postprocessor class.
Code lines: 3
Contained by: file stellar_populations.spectra.postprocess.Meiksin2006.F90

file: stellar_populations.spectra.postprocess.builder.lookup.F90
Description: Implements a stellar population spectra postprocessor builder which simply looks up postprocessors by name.
Code lines: 121

function: lookupbuild
Description: Return a stellar population spectra postprocessor by lookup via name.
Code lines: 15
Contained by: file stellar_populations.spectra.postprocess.builder.lookup.F90
Modules used: galacticus_error

function: lookupconstructorinternal
Description: Internal constructor for the lookup stellar population spectra postprocessor builder.
Code lines: 15
Contained by: file stellar_populations.spectra.postprocess.builder.lookup.F90
Modules used: galacticus_error

function: lookupconstructorparameters
Description: Constructor for the lookup stellar population spectra postprocessor builder class which takes a parameter list as input.
Code lines: 32
Contained by: file stellar_populations.spectra.postprocess.builder.lookup.F90
Modules used: galacticus_error input_parameters

subroutine: lookupdestructor
Description: Destructor for the lookup stellar population spectra postprocessor builder.
Code lines: 10
Contained by: file stellar_populations.spectra.postprocess.builder.lookup.F90

interface: stellarpopulationspectrapostprocessorbuilderlookup
Description: Constructors for the lookup stellar population spectra postprocessor builder class.
Code lines: 4
Contained by: file stellar_populations.spectra.postprocess.builder.lookup.F90

file: stellar_populations.spectra.postprocess.identity.F90
Description: An implementation of a spectrum postprocessor that does nothing.
Code lines: 61

function: identityconstructorparameters
Description: Constructor for the identity stellar population spectra postprocessor class which takes a parameter list as input.
Code lines: 11

Contained by: file stellar_populations.spectra.postprocess.identity.F90
Modules used: input_parameters

function: identitymultiplier
Description: Perform an identity postprocessing on a spectrum.
Code lines: 9
Contained by: file stellar_populations.spectra.postprocess.identity.F90

interface: stellarpopulationspectrapostprocessoridentity
Description: Constructors for the identity stellar population spectra postprocessor class.
Code lines: 3
Contained by: file stellar_populations.spectra.postprocess.identity.F90

file: stellar_populations.spectra.postprocess.recent.F90
Description: An implementation of a spectrum postprocessor that keeps only recent populations.
Code lines: 84

function: recentconstructorinternal
Description: Generic constructor for the recent spectrum postprocessor class.
Code lines: 8
Contained by: file stellar_populations.spectra.postprocess.recent.F90

function: recentconstructorparameters
Description: Default constructor for the recent spectrum postprocessor class.
Code lines: 18
Contained by: file stellar_populations.spectra.postprocess.recent.F90
Modules used: input_parameters

function: recentmultiplier
Description: Perform a recent postprocessing on a spectrum.
Code lines: 13
Contained by: file stellar_populations.spectra.postprocess.recent.F90

interface: stellarpopulationspectrapostprocessorrecent
Description: Constructors for the recent spectrum postprocessor class.
Code lines: 4
Contained by: file stellar_populations.spectra.postprocess.recent.F90

file: stellar_populations.spectra.postprocess.sequence.F90
Description: Implements a stellar population spectra postprocessor class which applies a sequence of other postprocessors.
Code lines: 174

type: postprocessorlist
Code lines: 3
Contained by: file stellar_populations.spectra.postprocess.sequence.F90

function: sequenceconstructorinternal
Description: Internal constructor for the sequence mstellar population spectra postprocessor class.

Code lines: 14
Contained by: file stellar_populations.spectra.postprocess.sequence.F90

function: sequenceconstructorparameters
Description: Constructor for the "sequence" stellar population spectra postprocessor class which takes a parameter set as input.
Code lines: 22
Contained by: file stellar_populations.spectra.postprocess.sequence.F90
Modules used: input_parameters

subroutine: sequencedeepcopy
Description: Perform a deep copy for the sequence stellar population spectra postprocessor class.
Code lines: 31
Contained by: file stellar_populations.spectra.postprocess.sequence.F90
Modules used: galacticus_error

subroutine: sequencedescriptor
Description: Return an input parameter list descriptor which could be used to recreate this object.
Code lines: 18
Contained by: file stellar_populations.spectra.postprocess.sequence.F90
Modules used: input_parameters

subroutine: sequencedestructor
Description: Destructor for the sequence stellar population spectra postprocessor class.
Code lines: 16
Contained by: file stellar_populations.spectra.postprocess.sequence.F90

function: sequencemultiplier
Description: Implement an sequence stellar population spectra postprocessor.
Code lines: 14
Contained by: file stellar_populations.spectra.postprocess.sequence.F90

interface: stellarpopulationspectrapostprocessorsequence
Description: Constructors for the "sequence" stellar population spectra postprocessor class.
Code lines: 4
Contained by: file stellar_populations.spectra.postprocess.sequence.F90

file: stellar_populations.spectra.postprocess.unescaped.F90
Description: An implementation of a spectrum postprocessor that keeps only unescaped populations.
Code lines: 80

interface: stellarpopulationspectrapostprocessorunescaped
Description: Constructors for the unescaped spectrum postprocessor class.
Code lines: 4
Contained by: file stellar_populations.spectra.postprocess.unescaped.F90

function: unescapedconstructorinternal
Description: Generic constructor for the unescaped spectrum postprocessor class.
Code lines: 8

Contained by: file stellar_populations.spectra.postprocess.unescaped.F90

function: unescapedconstructorparameters
Description: Constructor for the unescaped spectrum postprocessor class which accepts a parameter set as input.
Code lines: 18
Contained by: file stellar_populations.spectra.postprocess.unescaped.F90
Modules used: input_parameters

function: unescapedmultiplier
Description: Perform an unescaped postprocessing on a spectrum.
Code lines: 9
Contained by: file stellar_populations.spectra.postprocess.unescaped.F90

file: stellar_populations.standard.F90
Description: Implements a standard stellar population class.
Code lines: 648
Modules used: fgsl, stellar_astrophysics, stellar_feedback, stellar_population_spectra, stellar_populations_initial_mass_functions, supernovae_type_ia

interface: populationtable
Description: Constructors for the populationTable class.
Code lines: 3
Contained by: file stellar_populations.standard.F90

function: populationtableconstructor
Description: Constructor for the standard stellar population class which takes a parameter list as input.
Code lines: 13
Contained by: file stellar_populations.standard.F90

function: standardconstructorinternal
Description: Internal constructor for the standard stellar population.
Code lines: 39
Contained by: file stellar_populations.standard.F90

function: standardconstructorparameters
Description: Constructor for the standard stellar population class which takes a parameter list as input.
Code lines: 84
Contained by: file stellar_populations.standard.F90
Modules used: input_parameters

subroutine: standarddestructor
Description: Destructor for the standard stellar population class.
Code lines: 11
Contained by: file stellar_populations.standard.F90

function: standardintegrandenergyoutput

Description: Integrand used in evaluating energy output.
Code lines: 21
Contained by: file stellar_populations.standard.F90

function: standardintegrandrecycledfraction
Description: Integrand used in evaluating recycled fractions.
Code lines: 11
Contained by: file stellar_populations.standard.F90

function: standardintegrandyield
Description: Integrand used in evaluating metal yields.
Code lines: 37
Contained by: file stellar_populations.standard.F90

function: standardinterpolate
Description: Return the rate at which at cumulative property is being produced from this stellar population. The cumulative property is computed on a grid of age and metallicity. This is stored to file and will be read back in on subsequent runs. This is useful as computation of the table is relatively slow.
Code lines: 193
Contained by: file stellar_populations.standard.F90
Modules used: dates_and_times, file_utilities, galacticus_display, galacticus_error, galacticus_paths, input_parameters, io_hdf5, iso_c_binding, memory_management, numerical_constants_astronomical, numerical_integration2, numerical_interpolation, numerical_ranges, system_command, table_labels

function: standardrateenergy
Description: Return the rate at which energy is being output by this stellar population in $(\mathrm{km/s})^2$ Gyr^{-1}. The mean energy output rate per Gyr is computed between the given ageMinimum and ageMaximum (in Gyr).
Code lines: 11
Contained by: file stellar_populations.standard.F90

function: standardraterecycling
Description: Return the rate at which mass is being recycled from this stellar population. The mean recycling rate (i.e. the fraction of the population's mass returned to the ISM per Gyr) is computed between the given ageMinimum and ageMaximum (in Gyr).
Code lines: 11
Contained by: file stellar_populations.standard.F90

function: standardrateyield
Description: Return the rate at which mass is being recycled from this stellar population. The mean recycling rate (i.e. the fraction of the population's mass returned to the ISM per Gyr) is computed between the given ageMinimum and ageMaximum (in Gyr).
Code lines: 14
Contained by: file stellar_populations.standard.F90

function: `standardrecycledfractioninstantaneous`
Description: Return the recycled fraction from the stellar population in the instantaneous approximation.
Code lines: 14
Contained by: file stellar_populations.standard.F90
Modules used: numerical_constants_astronomical

function: `standardspectra`
Description: Return the stellar spectra associated with this population.
Code lines: 8
Contained by: file stellar_populations.standard.F90

function: `standardstarisevolved`
Description: Returns true if the specified star is evolved by the given `age`.
Code lines: 14
Contained by: file stellar_populations.standard.F90

function: `standardyieldinstantaneous`
Description: Return the metal yield from the stellar population in the instantaneous approximation.
Code lines: 14
Contained by: file stellar_populations.standard.F90
Modules used: numerical_constants_astronomical

interface: `stellarpopulationstandard`
Description: Constructors for the `standard` stellar population class.
Code lines: 4
Contained by: file stellar_populations.standard.F90

file: `structure_formation.cosmological_density_field.F90`
Description: Contains a module which provides a class that implements critical overdensity.
Code lines: 305

module: `cosmological_density_field`
Description: Provides an object that implements critical overdensities and halo environments.
Code lines: 283
Contained by: file structure_formation.cosmological_density_field.F90
Modules used: cosmology_functions, galacticus_nodes
Used by: subroutine dark_matter_halo_-correa2015_fit_parameters; file dark_matter_halos.mass_accretion_-history.Correa2015.F90; file dark_matter_halos.mass_accretion_-history.Wechsler2002.F90; file dark_matter_halos.mass_accretion_-history.Zhao2009.F90; file dark_matter_-profiles.structure.concentration.Bullock2001.F90; file dark_matter_-profiles.structure.concentration.Correa2015.F90; file dark_matter_-profiles.structure.concentration.Diemer-Kravtsov2014.F90; file dark_matter_-profiles.structure.concentration.Klypin2015.F90; file dark_matter_-profiles.structure.concentration.Ludlow2016-fit.F90; file dark_matter_-profiles.structure.concentration.NFW.F90

file dark_matter_profiles.structure.concentration.Prada2011.F90
file dark_matter_profiles.structure.concentration.Schneider2015.F90
file dark_matter_profiles.structure.shape.Gao2008.F90
file dark_matter_profiles.structure.shape.Klypin2015.F90
subroutine galacticus_function_classes_destroy
subroutine galacticus_state_retrieve
subroutine galacticus_state_store
file merger_trees.branching_probability.Parkinson_Cole_Helly.F90
file merger_trees.branching_probability.generalized_Press_Schechter.F90
subroutine generalizedpressschechtercomputecommonfactors
file merger_trees.construct.build.masses.distributions.mass_function.F90
file merger_trees.construct.builder.Cole2000.F90
file merger_trees.construct.read.importer.SussingMergerTrees.nbody.F90
file merger_trees.construct.read.importer.galacticus.F90
file merger_trees.operators.export.F90
file merger_trees.operators.regrid_times.F90
file models.likelihoods.halo_mass_function.F90
file nodes.property_extractor.halo_environment.F90
file satellites.merging.virial_orbits.Wetzel2010.F90
file structure_formation.excursion_sets.barrier.critical_overdensity.F90
file structure_formation.halo_bias.Press-Schechter.F90
file structure_formation.halo_bias.Sheth2001.F90
file structure_formation.halo_bias.Tinker2010.F90
file structure_formation.halo_mass_function.Bhattacharya2011.F90
file structure_formation.halo_mass_function.Despali_2015.F90
file structure_formation.halo_mass_function.Press-Schechter.F90
file structure_formation.halo_mass_function.Sheth-Tormen.F90
file structure_formation.halo_mass_function.Tinker2008Form.F90
file structure_formation.halo_mass_function.environment_averaged.F90
file structure_formation.halo_mass_function.environmental.F90
file structure_formation.power_spectrum.nonlinear.CosmicEmu.F90
file structure_formation.power_spectrum.standard.F90
file tasks.excursion_sets.F90
file tasks.halo_mass_function.F90
file tasks.merger_tree_file_builder.F90
file tasks.power_spectrum.F90
program test_diemerkravtsov2014_concentration
program tests_halo_mass_function_tinker
program test_parameters
program tests_power_spectrum
program tests_sigma
program tests_spherical_collapse_dark_energy_eds
program tests_spherical_collapse_dark_energy_omega_half
program tests_spherical_collapse_dark_energy_omega_two_thirds
program tests_spherical_collapse_dark_energy_lambda
program tests_spherical_collapse_dark_energy_open

file universe.operators.intergalactic_-
medium_state_evolve.F90

function: collapsetimeroot
Description: Function used in root finding for the collapse time at a given critical overdensity.
Code lines: 19
Contained by: module cosmological_density_field

function: collapsingmassroot
Description: Function used in root finding for the collapsing mass at a given time.
Code lines: 11
Contained by: module cosmological_density_field

file: structure_formation.cosmological_mass_variance.filtered_power_spectrum.F90
Description: An implementation of cosmological density field mass variance computed using a filtered power spectrum.
Code lines: 452
Modules used: cosmology_parameters, power_spectra_primordial_transferred, power_spectrum_window_functions, tables

interface: cosmologicalmassvariancefilteredpower
Description: Constructors for the filteredPower cosmological mass variance class.
Code lines: 4
Contained by: file structure_formation.cosmological_mass_variance.filtered_power_spectrum.F90

function: filteredpowerconstructorinternal
Description: Internal constructor for the filteredPower linear growth class.
Code lines: 18
Contained by: file structure_formation.cosmological_mass_variance.filtered_power_spectrum.F90

function: filteredpowerconstructorparameters
Description: Constructor for the filteredPower cosmological mass variance class which takes a parameter set as input.
Code lines: 57
Contained by: file structure_formation.cosmological_mass_variance.filtered_power_spectrum.F90
Modules used: input_parameters

subroutine: filteredpowerdestructor
Description: Destructor for the filteredPower linear growth class.
Code lines: 13
Contained by: file structure_formation.cosmological_mass_variance.filtered_power_spectrum.F90

function: filteredpowermass
Description: Return the mass corrresponding to the given root-variance of the cosmological density field.
Code lines: 34
Contained by: file structure_formation.cosmological_mass_variance.filtered_power_spectrum.F90

function: filteredpowerpowernormalization
Description: Return the normalization of the power spectrum.

Code lines: 8
Contained by: file structure_formation.cosmological_mass_variance.filtered_power_spectrum.F90

subroutine: filteredpowerretabulate
Description: Tabulate the cosmological mass variance.
Code lines: 143
Contained by: file structure_formation.cosmological_mass_variance.filtered_power_spectrum.F90
Modules used: galacticus_error numerical_constants_math

function: rootvariance
Description: Compute the root-variance of mass in spheres enclosing the given mass from the power spectrum.
Code lines: 24
Contained by: subroutine filteredpowerretabulate
Modules used: fgsl iso_c_binding numerical_constants_math numerical_integration

function: varianceintegrand
Description: Integrand function used in compute the variance in (real space) top-hat spheres from the power spectrum.
Code lines: 9
Contained by: subroutine filteredpowerretabulate

function: varianceintegrandtophat
Description: Integrand function used in compute the variance in (real space) top-hat spheres from the power spectrum.
Code lines: 9
Contained by: subroutine filteredpowerretabulate

function: filteredpowerrootvariance
Description: Return the root-variance of the cosmological density field in a spherical region containing the given mass on average.
Code lines: 10
Contained by: file structure_formation.cosmological_mass_variance.filtered_power_spectrum.F90

subroutine: filteredpowerrootvarianceandlogarithmicgradient
Description: Return the value and logairhtmic gradient with respect to mass of the root-variance of the cosmological density field in a spherical region containing the given mass on average.
Code lines: 21
Contained by: file structure_formation.cosmological_mass_variance.filtered_power_spectrum.F90
Modules used: numerical_constants_math

function: filteredpowerrootvariancelogarithmicgradient
Description: Return the logairhtmic gradient with respect to mass of the root-variance of the cosmological density field in a spherical region containing the given mass on average.
Code lines: 19
Contained by: file structure_formation.cosmological_mass_variance.filtered_power_spectrum.F90
Modules used: numerical_constants_math

function: filteredpowersigma8

Description: Return the value of σ_8.
Code lines: 7
Contained by: file structure_formation.cosmological_mass_variance.filtered_power_spectrum.F90

file: structure_formation.cosmological_mass_variance.peak_background_split.F90
Description: An implementation of cosmological density field mass variance which modifies another member of the class by offsetting for the peak-background split.
Code lines: 199
Modules used: cosmology_parameters

interface: cosmologicalmassvariancepeakbackgroundsplit
Description: Constructors for the peakBackgroundSplit cosmological mass variance class.
Code lines: 4
Contained by: file structure_formation.cosmological_mass_variance.peak_background_split.F90

function: variancepeakbackgroundsplitconstructorinternal
Description: Internal constructor for the peakBackgroundSplit linear growth class.
Code lines: 17
Contained by: file structure_formation.cosmological_mass_variance.peak_background_split.F90

function: variancepeakbackgroundsplitconstructorparameters
Description: Constructor for the peakBackgroundSplit cosmological mass variance class which takes a parameter set as input.
Code lines: 20
Contained by: file structure_formation.cosmological_mass_variance.peak_background_split.F90
Modules used: input_parameters

subroutine: variancepeakbackgroundsplitdestructor
Description: Destructor for the peakBackgroundSplit linear growth class.
Code lines: 9
Contained by: file structure_formation.cosmological_mass_variance.peak_background_split.F90

function: variancepeakbackgroundsplitmass
Description: Return the mass corrresponding to the given root-variance of the cosmological density field.
Code lines: 8
Contained by: file structure_formation.cosmological_mass_variance.peak_background_split.F90

function: variancepeakbackgroundsplitpowernormalization
Description: Return the normalization of the power spectrum.
Code lines: 10
Contained by: file structure_formation.cosmological_mass_variance.peak_background_split.F90
Modules used: galacticus_error

function: variancepeakbackgroundsplitrootvariance
Description: Return the root-variance of the cosmological density field in a spherical region containing the given mass on average.
Code lines: 15
Contained by: file structure_formation.cosmological_mass_variance.peak_background_split.F90

subroutine: variancepeakbackgroundsplitrootvarianceandlogarithmicgradient

Description: Return the value and logarithmic gradient with respect to mass of the root-variance of the cosmological density field in a spherical region containing the given `mass` on average.
Code lines: 19
Contained by: file `structure_formation.cosmological_mass_variance.peak_background_split.F90`
Modules used: `numerical_constants_math`

function: `variancepeakbackgroundsplitrootvariancelogarithmicgradient`
Description: Return the logairhtmic gradient with respect to mass of the root-variance of the cosmological density field in a spherical region containing the given `mass` on average.
Code lines: 16
Contained by: file `structure_formation.cosmological_mass_variance.peak_background_split.F90`
Modules used: `numerical_constants_math`

function: `variancepeakbackgroundsplitsigma8`
Description: Return the value of σ_8.
Code lines: 10
Contained by: file `structure_formation.cosmological_mass_variance.peak_background_split.F90`
Modules used: `numerical_constants_math`

file: `structure_formation.critical_overdensity.Kitayama_Suto1996.F90`
Description: Contains a module which implements a critical overdensity class based on the fitting functions of Kitayama and Suto [1996].
Code lines: 170
Modules used: `cosmology_functions` `dark_matter_particles` `linear_growth`

interface: `criticaloverdensitykitayamasuto1996`
Description: Constructors for the `kitayamaSuto1996` critical overdensity class.
Code lines: 4
Contained by: file `structure_formation.critical_overdensity.Kitayama_Suto1996.F90`

function: `kitayamasuto1996constructorinternal`
Description: Internal constructor for the `kitayamaSuto1996` critical overdensity class.
Code lines: 21
Contained by: file `structure_formation.critical_overdensity.Kitayama_Suto1996.F90`
Modules used: `dark_matter_particles` `galacticus_error`

function: `kitayamasuto1996constructorparameters`
Description: Constructor for the `kitayamaSuto1996` critical overdensity class which takes a parameter set as input.
Code lines: 24
Contained by: file `structure_formation.critical_overdensity.Kitayama_Suto1996.F90`
Modules used: `galacticus_error` `input_parameters`

subroutine: `kitayamasuto1996destructor`
Description: Destructor for the `kitayamaSuto1996` critical overdensity class.
Code lines: 10
Contained by: file `structure_formation.critical_overdensity.Kitayama_Suto1996.F90`

function: `kitayamasuto1996gradientmass`
Description: Return the gradient with respect to mass of critical overdensity at the given time and mass.
Code lines: 12
Contained by: file `structure_formation.critical_overdensity.Kitayama_Suto1996.F90`

function: `kitayamasuto1996gradienttime`
Description: Return the gradient with respect to time of critical overdensity at the given time and mass.
Code lines: 15
Contained by: file `structure_formation.critical_overdensity.Kitayama_Suto1996.F90`
Modules used: `numerical_constants_math`

function: `kitayamasuto1996ismassdependent`
Description: Return whether the critical overdensity is mass dependent.
Code lines: 8
Contained by: file `structure_formation.critical_overdensity.Kitayama_Suto1996.F90`

function: `kitayamasuto1996value`
Description: Return the critical overdensity at the given time and mass.
Code lines: 16
Contained by: file `structure_formation.critical_overdensity.Kitayama_Suto1996.F90`
Modules used: `numerical_constants_math`

file: `structure_formation.critical_overdensity.environmental.F90`
Description: Contains a module which implements an environmental critical overdensity class.
Code lines: 184
Modules used: `linear_growth`

interface: `criticaloverdensityenvironmental`
Description: Constructors for the `environmental` critical overdensity class.
Code lines: 4
Contained by: file `structure_formation.critical_overdensity.environmental.F90`

function: `environmentalconstructorinternal`
Description: Internal constructor for the `environmental` critical overdensity class.
Code lines: 14
Contained by: file `structure_formation.critical_overdensity.environmental.F90`

function: `environmentalconstructorparameters`
Description: Constructor for the `environmental` critical overdensity class which takes a parameter set as input.
Code lines: 35
Contained by: file `structure_formation.critical_overdensity.environmental.F90`
Modules used: `input_parameters`

subroutine: `environmentaldestructor`
Description: Destructor for the `environmental` critical overdensity class.
Code lines: 11
Contained by: file `structure_formation.critical_overdensity.environmental.F90`

function: `environmentalgradientmass`
Description: Return the gradient with respect to mass of critical overdensity at the given time and mass.
Code lines: 18
Contained by: file `structure_formation.critical_overdensity.environmental.F90`
Modules used: `galacticus_nodes`

function: `environmentalgradienttime`
Description: Return the gradient with respect to time of critical overdensity at the given time and mass.
Code lines: 19
Contained by: file `structure_formation.critical_overdensity.environmental.F90`
Modules used: `galacticus_nodes`

function: `environmentalismassdependent`
Description: Return whether the critical overdensity is mass dependent.
Code lines: 7
Contained by: file `structure_formation.critical_overdensity.environmental.F90`

function: `environmentalvalue`
Description: Return the critical overdensity for collapse at the given time and mass.
Code lines: 18
Contained by: file `structure_formation.critical_overdensity.environmental.F90`
Modules used: `galacticus_nodes`

file: `structure_formation.critical_overdensity.fixed.F90`
Description: Contains a module which implements an fixed critical overdensity class.
Code lines: 159
Modules used: `cosmology_functions` `linear_growth`

interface: `criticaloverdensityfixed`
Description: Constructors for the fixed critical overdensity class.
Code lines: 4
Contained by: file `structure_formation.critical_overdensity.fixed.F90`

function: `fixedconstructorinternal`
Description: Internal constructor for the fixed critical overdensity class.
Code lines: 11
Contained by: file `structure_formation.critical_overdensity.fixed.F90`

function: `fixedconstructorparameters`
Description: Constructor for the fixed critical overdensity class which takes a parameter set as input.
Code lines: 31
Contained by: file `structure_formation.critical_overdensity.fixed.F90`
Modules used: `input_parameters` `numerical_constants_math`

subroutine: `fixeddestructor`
Description: Destructor for the fixed critical overdensity class.
Code lines: 9
Contained by: file `structure_formation.critical_overdensity.fixed.F90`

function: `fixedgradientmass`
Description: Return the gradient with respect to mass of critical overdensity at the given time and mass.
Code lines: 12
Contained by: file `structure_formation.critical_overdensity.fixed.F90`

function: `fixedgradienttime`
Description: Return the gradient with respect to time of critical overdensity at the given time and mass.
Code lines: 14
Contained by: file `structure_formation.critical_overdensity.fixed.F90`

function: `fixedismassdependent`
Description: Return whether the critical overdensity is mass dependent.
Code lines: 8
Contained by: file `structure_formation.critical_overdensity.fixed.F90`

function: `fixedvalue`
Description: Return the critical overdensity at the given time and mass.
Code lines: 14
Contained by: file `structure_formation.critical_overdensity.fixed.F90`

file: `structure_formation.critical_overdensity.peak_background_split.F90`
Description: Contains a module which implements an peak-background split critical overdensity class.
Code lines: 149

interface: `criticaloverdensitypeakbackgroundsplit`
Description: Constructors for the `peakBackgroundSplit` critical overdensity class.
Code lines: 4
Contained by: file `structure_formation.critical_overdensity.peak_background_split.F90`

function: `peakbackgroundsplitconstructorinternal`
Description: Internal constructor for the `peakBackgroundSplit` critical overdensity class.
Code lines: 11
Contained by: file `structure_formation.critical_overdensity.peak_background_split.F90`

function: `peakbackgroundsplitconstructorparameters`
Description: Constructor for the `peakBackgroundSplit` critical overdensity class which takes a parameter set as input.
Code lines: 23
Contained by: file `structure_formation.critical_overdensity.peak_background_split.F90`
Modules used: `input_parameters`

subroutine: `peakbackgroundsplitdestructor`
Description: Destructor for the `peakBackgroundSplit` critical overdensity class.
Code lines: 10
Contained by: file `structure_formation.critical_overdensity.peak_background_split.F90`

function: `peakbackgroundsplitgradientmass`
Description: Return the gradient with respect to mass of critical overdensity at the given time and mass.

Code lines: 11
Contained by: file structure_formation.critical_overdensity.peak_background_split.F90

function: peakbackgroundsplitgradienttime
Description: Return the gradient with respect to time of critical overdensity at the given time and mass.
Code lines: 11
Contained by: file structure_formation.critical_overdensity.peak_background_split.F90

function: peakbackgroundsplitismassdependent
Description: Return whether the critical overdensity is mass dependent.
Code lines: 7
Contained by: file structure_formation.critical_overdensity.peak_background_split.F90

function: peakbackgroundsplitvalue
Description: Return the critical overdensity for collapse at the given time and mass.
Code lines: 18
Contained by: file structure_formation.critical_overdensity.peak_background_split.F90

file: structure_formation.critical_overdensity.spherical_collapse_matter_dark_energy.F90
Description: An implementation of critical overdensity for collapse based on spherical collapse in a matter plus dark energy universe.
Code lines: 123

interface: criticaloverdensitysphericalcollapsematterde
Description: Constructors for the sphericalCollapseMatterDE critical overdensity for collapse class.
Code lines: 4
Contained by: file structure_formation.critical_overdensity.spherical_collapse_matter_dark_energy.F90

function: sphericalcollapsematterdeconstructorinternal
Description: Internal constructor for the sphericalCollapseMatterDE critical overdensity class.
Code lines: 23
Contained by: file structure_formation.critical_overdensity.spherical_collapse_matter_dark_energy.F90
Modules used: dark_matter_particles galacticus_error

function: sphericalcollapsematterdeconstructorparameters
Description: Constructor for the sphericalCollapseMatterDE critical overdensity class which takes a parameter set as input.
Code lines: 34
Contained by: file structure_formation.critical_overdensity.spherical_collapse_matter_dark_energy.F90
Modules used: dark_matter_particles galacticus_error input_parameters

subroutine: sphericalcollapsematterderetabulate
Description: Recompute the look-up tables for critical overdensity for collapse.
Code lines: 21
Contained by: file structure_formation.critical_overdensity.spherical_collapse_matter_dark_energy.F90
Modules used: spherical_collapse_matter_dark_energy

file: structure_formation.critical_overdensity.spherical_collapse_matter_lambda.F90

Description: An implementation of critical overdensity for collapse based on spherical collapse in a matter plus cosmological constant universe.
Code lines: 239
Modules used: cosmology_functions, dark_matter_particles, linear_growth, tables

interface: criticaloverdensitysphericalcollapsematterlambda
Description: Constructors for the sphericalCollapseMatterLambda critical overdensity for collapse class.
Code lines: 4
Contained by: file structure_formation.critical_overdensity.spherical_collapse_matter_lambda.F90

function: sphericalcollapsematterlambdaconstructorinternal
Description: Internal constructor for the sphericalCollapseMatterLambda critical overdensity class.
Code lines: 24
Contained by: file structure_formation.critical_overdensity.spherical_collapse_matter_lambda.F90
Modules used: dark_matter_particles, galacticus_error

function: sphericalcollapsematterlambdaconstructorparameters
Description: Constructor for the sphericalCollapseMatterLambda critical overdensity class which takes a parameter set as input.
Code lines: 41
Contained by: file structure_formation.critical_overdensity.spherical_collapse_matter_lambda.F90
Modules used: input_parameters

subroutine: sphericalcollapsematterlambdadestructor
Description: Destructor for the sphericalCollapseMatterLambda critical overdensity for collapse class.
Code lines: 14
Contained by: file structure_formation.critical_overdensity.spherical_collapse_matter_lambda.F90

function: sphericalcollapsematterlambdagradientmass
Description: Return the gradient with respect to mass of critical overdensity at the given time and mass.
Code lines: 14
Contained by: file structure_formation.critical_overdensity.spherical_collapse_matter_lambda.F90
Modules used: cosmology_functions, linear_growth

function: sphericalcollapsematterlambdagradienttime
Description: Return the time derivative of the critical overdensity at the given epoch, based spherical collapse in a matter plus cosmological constant universe.
Code lines: 18
Contained by: file structure_formation.critical_overdensity.spherical_collapse_matter_lambda.F90

function: sphericalcollapsematterlambdaismassdependent
Description: Return whether the critical overdensity is mass dependent.
Code lines: 8
Contained by: file structure_formation.critical_overdensity.spherical_collapse_matter_lambda.F90

subroutine: sphericalcollapsematterlambdaretabulate
Description: Recompute the look-up tables for critical overdensity for collapse.

Code lines: 21
Contained by: file structure_formation.critical_overdensity.spherical_collapse_matter_lambda.F90
Modules used: spherical_collapse_matter_lambda

function: sphericalcollapsematterlambdavalue
Description: Return the critical overdensity at the given epoch, based spherical collapse in a matter plus cosmological constant universe.
Code lines: 18
Contained by: file structure_formation.critical_overdensity.spherical_collapse_matter_lambda.F90
Modules used: galacticus_error

file: structure_formation.critical_overdensity.warm_dark_matter.F90
Description: Contains a module which implements a critical overdensity for collapse the WDM modifier of Barkana et al. [2001].
Code lines: 279
Modules used: cosmology_parameters dark_matter_particles fgsl

function: barkana2001wdmconstructorinternal
Description: Internal constructor for the "barkana2001WDM" critical overdensity for collapse class.
Code lines: 44
Contained by: file structure_formation.critical_overdensity.warm_dark_matter.F90
Modules used: fox_dom galacticus_error galacticus_paths io_xml iso_varying_string

function: barkana2001wdmconstructorparameters
Description: Constructor for the "barkana2001WDM" critical overdensity for collapse class which takes a parameter set as input.
Code lines: 35
Contained by: file structure_formation.critical_overdensity.warm_dark_matter.F90
Modules used: input_parameters

subroutine: barkana2001wdmdestructor
Description: Destructor for the barkana2001WDM critical overdensity for collapse class.
Code lines: 11
Contained by: file structure_formation.critical_overdensity.warm_dark_matter.F90

function: barkana2001wdmgradientmass
Description: Return the gradient with respect to mass of critical overdensity at the given time and mass.
Code lines: 40
Contained by: file structure_formation.critical_overdensity.warm_dark_matter.F90
Modules used: fgsl galacticus_error numerical_interpolation table_labels

function: barkana2001wdmgradienttime
Description: Return the gradient with respect to time of critical overdensity at the given time and mass.
Code lines: 12
Contained by: file structure_formation.critical_overdensity.warm_dark_matter.F90

function: barkana2001wdmismassdependent
Description: Return whether the critical overdensity is mass dependent.
Code lines: 8
Contained by: file structure_formation.critical_overdensity.warm_dark_matter.F90

function: barkana2001wdmvalue
Description: Returns a mass scaling for critical overdensities based on the results of Barkana et al. [2001]. This method assumes that their results for the original collapse barrier (i.e. the critical overdensity, and which they call B_0) scale with the effective Jeans mass of the warm dark matter particle as computed using their eqn. (10).
Code lines: 51
Contained by: file structure_formation.critical_overdensity.warm_dark_matter.F90
Modules used: fgsl, galacticus_error, numerical_interpolation, table_labels

interface: criticaloverdensitybarkana2001wdm
Description: Constructors for the "barkana2001WDM" critical overdensity for collapse class.
Code lines: 4
Contained by: file structure_formation.critical_overdensity.warm_dark_matter.F90

file: structure_formation.excursion_sets.barrier.F90
Description: Contains a module which provides a class that implements barriers for the excursion set problem.
Code lines: 61

module: excursion_sets_barriers
Description: Provides a class that implements barriers for the excursion set problem.
Code lines: 39
Contained by: file structure_formation.excursion_sets.barrier.F90
Modules used: galacticus_nodes
Used by: subroutine galacticus_function_classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; file structure_formation.excursion_sets.first_crossing_distribution.Farahi.F90; file structure_formation.excursion_sets.first_crossing_distribution.Zhang_Hui.F90; file structure_formation.excursion_sets.first_crossing_distribution.linear_barrier.F90; file tasks.excursion_sets.F90

file: structure_formation.excursion_sets.barrier.critical_overdensity.F90
Description: Contains a module which implements a critical overdensity excursion set barrier class.
Code lines: 130
Modules used: cosmological_density_field

function: criticaloverdensitybarrier
Description: Return the excursion set barrier at the given variance and time.
Code lines: 20

Contained by: file structure_formation.excursion_sets.barrier.critical_overdensity.F90

function: criticaloverdensitybarriergradient
Description: Return the gradient with respect to variance of the excursion set barrier at the given variance and time.
Code lines: 22
Contained by: file structure_formation.excursion_sets.barrier.critical_overdensity.F90

function: criticaloverdensityconstructorinternal
Description: Internal constructor for the critical overdensity excursion set class.
Code lines: 9
Contained by: file structure_formation.excursion_sets.barrier.critical_overdensity.F90

function: criticaloverdensityconstructorparameters
Description: Constructor for the critical overdensity excursion set class which takes a parameter set as input.
Code lines: 17
Contained by: file structure_formation.excursion_sets.barrier.critical_overdensity.F90
Modules used: input_parameters

subroutine: criticaloverdensitydestructor
Description: Destructor for the critical overdensity excursion set barrier class.
Code lines: 8
Contained by: file structure_formation.excursion_sets.barrier.critical_overdensity.F90

interface: excursionsetbarriercriticaloverdensity
Description: Constructors for the critical overdensity excursion set barrier class.
Code lines: 4
Contained by: file structure_formation.excursion_sets.barrier.critical_overdensity.F90

file: structure_formation.excursion_sets.barrier.linear.F90
Description: Contains a module which implements a linear excursion set barrier class.
Code lines: 108

interface: excursionsetbarrierlinear
Description: Constructors for the linear excursion set barrier class.
Code lines: 4
Contained by: file structure_formation.excursion_sets.barrier.linear.F90

function: linearbarrier
Description: Return the excursion set barrier at the given variance and time.
Code lines: 11
Contained by: file structure_formation.excursion_sets.barrier.linear.F90

function: linearbarriergradient
Description: Return the gradient with respect to variance of the excursion set barrier at the given variance and time.
Code lines: 11
Contained by: file structure_formation.excursion_sets.barrier.linear.F90

function: `linearconstructorinternal`
Description: Internal constructor for the linear excursion set class.
Code lines: 8
Contained by: file `structure_formation.excursion_sets.barrier.linear.F90`

function: `linearconstructorparameters`
Description: Constructor for the linear excursion set class which takes a parameter set as input.
Code lines: 30
Contained by: file `structure_formation.excursion_sets.barrier.linear.F90`
Modules used: `input_parameters`

file: `structure_formation.excursion_sets.barrier.quadratic.F90`
Description: Contains a module which implements a quadratic excursion set barrier class.
Code lines: 117

interface: `excursionsetbarrierquadratic`
Description: Constructors for the quadratic excursion set barrier class.
Code lines: 4
Contained by: file `structure_formation.excursion_sets.barrier.quadratic.F90`

function: `quadraticbarrier`
Description: Return the excursion set barrier at the given variance and time.
Code lines: 11
Contained by: file `structure_formation.excursion_sets.barrier.quadratic.F90`

function: `quadraticbarriergradient`
Description: Return the gradient with respect to variance of the excursion set barrier at the given variance and time.
Code lines: 11
Contained by: file `structure_formation.excursion_sets.barrier.quadratic.F90`

function: `quadraticconstructorinternal`
Description: Internal constructor for the quadratic excursion set class.
Code lines: 8
Contained by: file `structure_formation.excursion_sets.barrier.quadratic.F90`

function: `quadraticconstructorparameters`
Description: Constructor for the quadratic excursion set class which takes a parameter set as input.
Code lines: 39
Contained by: file `structure_formation.excursion_sets.barrier.quadratic.F90`
Modules used: `input_parameters`

file: `structure_formation.excursion_sets.barrier.remap.Sheth-Mo-Tormen.F90`
Description: Contains a module which implements an excursion set barrier class which remaps another class using the Sheth et al. [2001] ellipsoidal collapse parameterization.
Code lines: 158

interface: `excursionsetbarrierremapshethmotormen`

Description: Constructors for the remap scale excursion set barrier class.
Code lines: 4
Contained by: file structure_formation.excursion_sets.barrier.remap.Sheth-Mo-Tormen.F90

function: remapshethmotormenbarrier
Description: Return the excursion set barrier at the given variance and time.
Code lines: 11
Contained by: file structure_formation.excursion_sets.barrier.remap.Sheth-Mo-Tormen.F90

function: remapshethmotormenbarriergradient
Description: Return the gradient with respect to variance of the excursion set barrier at the given variance and time.
Code lines: 21
Contained by: file structure_formation.excursion_sets.barrier.remap.Sheth-Mo-Tormen.F90

function: remapshethmotormenconstructorinternal
Description: Internal constructor for the critical overdensity excursion set class.
Code lines: 13
Contained by: file structure_formation.excursion_sets.barrier.remap.Sheth-Mo-Tormen.F90
Modules used: galacticus_error

function: remapshethmotormenconstructorparameters
Description: Constructor for the critical overdensity excursion set class which takes a parameter set as input.
Code lines: 51
Contained by: file structure_formation.excursion_sets.barrier.remap.Sheth-Mo-Tormen.F90
Modules used: input_parameters

subroutine: remapshethmotormendestructor
Description: Destructor for the critical overdensity excursion set barrier class.
Code lines: 8
Contained by: file structure_formation.excursion_sets.barrier.remap.Sheth-Mo-Tormen.F90

file: structure_formation.excursion_sets.barrier.remap.null.F90
Description: Contains a module which implements a null remapping of excursion set barriers.
Code lines: 89

module: excursion_sets_barriers_remap_null
Description: Implements a null remapping of excursion set barriers.
Code lines: 67
Contained by: file structure_formation.excursion_sets.barrier.remap.null.F90

subroutine: excursion_sets_barrier_gradient_remap_null
Description: Return the gradient of the barrier for excursion set calculations unmodified.
Code lines: 10
Contained by: module excursion_sets_barriers_remap_null

subroutine: excursion_sets_barrier_remap_null
Description: Return the barrier for excursion set calculations unmodified.

Code lines: 10
Contained by: module excursion_sets_barriers_remap_null

subroutine: excursion_sets_barriers_remap_null_initialize
Description: Initialize the null excursion set barrier remapping module.
Code lines: 22
Contained by: module excursion_sets_barriers_remap_null
Modules used: iso_varying_string

file: structure_formation.excursion_sets.barrier.remap.scale.F90
Description: Contains a module which implements an excursion set barrier class which remaps another class by multiplying by a constant.
Code lines: 129

interface: excursionsetbarrierremapscale
Description: Constructors for the remap scale excursion set barrier class.
Code lines: 4
Contained by: file structure_formation.excursion_sets.barrier.remap.scale.F90

function: remapscalebarrier
Description: Return the excursion set barrier at the given variance and time.
Code lines: 11
Contained by: file structure_formation.excursion_sets.barrier.remap.scale.F90

function: remapscalebarriergradient
Description: Return the gradient with respect to variance of the excursion set barrier at the given variance and time.
Code lines: 11
Contained by: file structure_formation.excursion_sets.barrier.remap.scale.F90

function: remapscaleconstructorinternal
Description: Internal constructor for the critical overdensity excursion set class.
Code lines: 13
Contained by: file structure_formation.excursion_sets.barrier.remap.scale.F90
Modules used: galacticus_error

function: remapscaleconstructorparameters
Description: Constructor for the critical overdensity excursion set class which takes a parameter set as input.
Code lines: 33
Contained by: file structure_formation.excursion_sets.barrier.remap.scale.F90
Modules used: input_parameters

subroutine: remapscaledestructor
Description: Destructor for the critical overdensity excursion set barrier class.
Code lines: 7
Contained by: file structure_formation.excursion_sets.barrier.remap.scale.F90

file: structure_formation.excursion_sets.first_crossing_distribution.F90

Description: Contains a module which provides a class for first crossing distributions for excursion set calculations.
Code lines: 66

module: `excursion_sets_first_crossings`
Description: Provides a class for first crossing distributions for excursion set calculations.
Code lines: 43
Contained by: file `structure_formation.excursion_sets.first_crossing_distribution.F90`
Modules used: `galacticus_nodes`
Used by: subroutine `galacticus_function_-classes_destroy`; subroutine `galacticus_state_retrieve`; subroutine `galacticus_state_store`; file `merger_trees.branching_-probability.generalized_Press_-Schechter.F90`; file `structure_formation.halo_mass_-function.Press-Schechter.F90`; file `tasks.excursion_sets.F90`

file: `structure_formation.excursion_sets.first_crossing_distribution.Farahi.F90`
Description: Contains a module which implements a excursion set first crossing statistics class using the algorithm of Benson et al. [2012].
Code lines: 968
Modules used: `cosmology_functions`; `excursion_sets_barriers`; `fgsl`; `file_utilities`; `iso_c_binding`

interface: `excursionsetfirstcrossingfarahi`
Description: Constructors for the Farahi excursion set barrier class.
Code lines: 4
Contained by: file `structure_formation.excursion_sets.first_crossing_distribution.Farahi.F90`

function: `farahiconstructorinternal`
Description: Internal constructor for the Farahi excursion set class first crossing class.
Code lines: 50
Contained by: file `structure_formation.excursion_sets.first_crossing_distribution.Farahi.F90`
Modules used: `file_utilities`; `galacticus_display`; `galacticus_paths`; `input_parameters`; `system_command`

function: `farahiconstructorparameters`
Description: Constructor for the Farahi excursion set class first crossing class which takes a parameter set as input.
Code lines: 34
Contained by: file `structure_formation.excursion_sets.first_crossing_distribution.Farahi.F90`
Modules used: `input_parameters`

subroutine: `farahidestructor`
Description: Destructor for the Farahi excursion set first crossing class.
Code lines: 14
Contained by: file `structure_formation.excursion_sets.first_crossing_distribution.Farahi.F90`

Modules used: numerical_interpolation

subroutine: farahifileread
Description: Read tabulated data on excursion set first crossing probabilities from file.
Code lines: 129
Contained by: file structure_formation.excursion_sets.first_crossing_distribution.Farahi.F90
Modules used: file_utilities, galacticus_display, io_hdf5, iso_varying_string, memory_management, numerical_interpolation

subroutine: farahifilewrite
Description: Write tabulated data on excursion set first crossing probabilities to file.
Code lines: 35
Contained by: file structure_formation.excursion_sets.first_crossing_distribution.Farahi.F90
Modules used: hdf5, io_hdf5

function: farahiprobability
Description: Return the excursion set barrier at the given variance and time.
Code lines: 191
Contained by: file structure_formation.excursion_sets.first_crossing_distribution.Farahi.F90
Modules used: error_functions, galacticus_display, kind_numbers, memory_management, mpi_utilities, numerical_interpolation, numerical_ranges

function: farahirate
Description: Return the excursion set barrier at the given variance and time.
Code lines: 56
Contained by: file structure_formation.excursion_sets.first_crossing_distribution.Farahi.F90
Modules used: numerical_interpolation

function: farahiratenoncrossing
Description: Return the rate for excursion set non-crossing.
Code lines: 31
Contained by: file structure_formation.excursion_sets.first_crossing_distribution.Farahi.F90
Modules used: numerical_interpolation

subroutine: farahiratetabulate
Description: Tabulate the excursion set crossing rate.
Code lines: 256
Contained by: file structure_formation.excursion_sets.first_crossing_distribution.Farahi.F90
Modules used: error_functions, galacticus_display, kind_numbers, memory_management, mpi_utilities, numerical_interpolation, numerical_ranges

function: farahivariancerange

Description: Builds a numerical range between `rangeMinimum` and `rangeMaximum` using `rangeNumber` points with spacing that varies from logarithmic to linear spacing with the transition point controlled by `exponent`. Specifically, suppose we have $N =$`rangeNumber` points in the range, from $S_{\min} =$`rangeMinimum` to $S_{\max} =$`rangeMaximum`. We define $f_i = (i-1)/(N-1)$ where i runs from 1 to N. We then define:

$$f_i = \frac{\int_{S_{\min}}^{S_i} x^{n_i} \mathrm{d}x}{\int_{S_{\min}}^{S_{\max}} x^{n_i} \mathrm{d}x}, \tag{18.52}$$

and solve for S_i to find the i^{th} range value. If $n_i = 0$ then this will give S_i linearly spaced between $S_{\min}$ and $S_{\max}$, while if $n_i = -1$ this will give S_i logarithmically spaced between $S_{\min}$ and $S_{\max}$. Therefore, if we make n_i vary from -1 to 0 at i ranges from 1 to N we will get a smooth transition from logarithmic to linear spacing. We choose to use $n_i = -1 + f_i^{\alpha}$ where $\alpha =$`exponent` is a supplied argument.

Code lines: 33

Contained by: file structure_formation.excursion_sets.first_crossing_distribution.Farahi.F90

file: structure_formation.excursion_sets.first_crossing_distribution.Farahi.midpoint.F90
Description: Contains a module which implements a excursion set first crossing statistics class using the algorithm of Benson et al. [2012], but using a midpoint method to perform the integrations [Du et al., 2017].
Code lines: 563

interface: excursionsetfirstcrossingfarahimidpoint
Description: Constructors for the Farahi-midpoint excursion set barrier class.
Code lines: 4
Contained by: file structure_formation.excursion_sets.first_crossing_distribution.Farahi.midpoint.F90

function: farahimidpointconstructorinternal
Description: Internal constructor for the Farahi-midpoint excursion set class first crossing class.
Code lines: 12
Contained by: file structure_formation.excursion_sets.first_crossing_distribution.Farahi.midpoint.F90
Modules used: input_parameters

function: farahimidpointconstructorparameters
Description: Constructor for the Farahi-midpoint excursion set class first crossing class which takes a parameter set as input.
Code lines: 9
Contained by: file structure_formation.excursion_sets.first_crossing_distribution.Farahi.midpoint.F90
Modules used: input_parameters

function: farahimidpointprobability
Description: Return the excursion set barrier at the given variance and time.
Code lines: 212
Contained by: file structure_formation.excursion_sets.first_crossing_distribution.Farahi.midpoint.F90
Modules used: error_functions, galacticus_display, kind_numbers, memory_management, mpi_utilities, numerical_interpolation, numerical_ranges

subroutine: farahimidpointratetabulate
Description: Tabulate the excursion set crossing rate.
Code lines: 281
Contained by: file structure_formation.excursion_sets.first_crossing_distribution.Farahi.midpoint.F90
Modules used: error_functions, galacticus_display, kind_numbers, memory_management, mpi_utilities, numerical_interpolation, numerical_ranges

file: structure_formation.excursion_sets.first_crossing_distribution.Zhang_Hui.F90
Description: Contains a module which implements a excursion set first crossing statistics class utilizing the algorithm of Zhang and Hui [2006].
Code lines: 364
Modules used: excursion_sets_barriers, fgsl

interface: `excursionsetfirstcrossingzhanghui`
Description: Constructors for the Zhang and Hui [2006] excursion set barrier class.
Code lines: 4
Contained by: file `structure_formation.excursion_sets.first_crossing_distribution.Zhang_Hui.F90`

function: `zhanghuiconstructorinternal`
Description: Constructor for the linear barrier excursion set class first crossing class which takes a parameter set as input.
Code lines: 18
Contained by: file `structure_formation.excursion_sets.first_crossing_distribution.Zhang_Hui.F90`
Modules used: `input_parameters`

function: `zhanghuiconstructorparameters`
Description: Constructor for the linear barrier excursion set class first crossing class which takes a parameter set as input.
Code lines: 13
Contained by: file `structure_formation.excursion_sets.first_crossing_distribution.Zhang_Hui.F90`
Modules used: `input_parameters`

function: `zhanghuidelta`
Description: Returns the factor $\Delta i, j$ in the Zhang and Hui [2006] algorithm for excursion set barrier crossing probabilities.
Code lines: 22
Contained by: file `structure_formation.excursion_sets.first_crossing_distribution.Zhang_Hui.F90`

subroutine: `zhanghuidestructor`
Description: Destructor for the critical overdensity excursion set barrier class.
Code lines: 7
Contained by: file `structure_formation.excursion_sets.first_crossing_distribution.Zhang_Hui.F90`

function: `zhanghuig1`
Description: Returns the function $g_1(S)$ in the Zhang and Hui [2006] algorithm for excursion set barrier crossing probabilities.
Code lines: 12
Contained by: file `structure_formation.excursion_sets.first_crossing_distribution.Zhang_Hui.F90`
Modules used: `math_distributions_gaussian`

function: `zhanghuig2`
Description: Returns the function $g_2(S, S')$ in the Zhang and Hui [2006] algorithm for excursion set barrier crossing probabilities.
Code lines: 20
Contained by: file `structure_formation.excursion_sets.first_crossing_distribution.Zhang_Hui.F90`
Modules used: `math_distributions_gaussian`

function: `zhanghuig2integrated`
Description: Integrated function $g_2(S, S')$ in the Zhang and Hui [2006] algorithm for excursion set barrier crossing probabilities.
Code lines: 50
Contained by: file `structure_formation.excursion_sets.first_crossing_distribution.Zhang_Hui.F90`

Modules used: fgsl numerical_comparison numerical_integration

function: zhanghuig2integrand
Description: Integrand function used in computing $\Delta_{i,i}$ in the Zhang and Hui [2006] algorithm for excursion set barrier crossing probabilities.
Code lines: 15
Contained by: function zhanghuig2integrated
Modules used: math_distributions_gaussian

function: zhanghuiprobability
Description: Return the excursion set barrier at the given variance and time.
Code lines: 84
Contained by: file structure_formation.excursion_sets.first_crossing_distribution.Zhang_Hui.F90
Modules used: galacticus_display iso_c_binding memory_management numerical_interpolation numerical_ranges

function: zhanghuirate
Description: Return the excursion set barrier at the given variance and time. This method is not implemented.
Code lines: 12
Contained by: file structure_formation.excursion_sets.first_crossing_distribution.Zhang_Hui.F90
Modules used: galacticus_error

function: zhanghuiratenoncrossing
Description: Return the rate for excursion set non-crossing. This method is not implemented.
Code lines: 12
Contained by: file structure_formation.excursion_sets.first_crossing_distribution.Zhang_Hui.F90
Modules used: galacticus_error

file: structure_formation.excursion_sets.first_crossing_distribution.Zhang_Hui_high_order.F90
Description: Contains a module which implements a excursion set first crossing statistics class utilizing a higher order generalization of the algorithm of Zhang and Hui [2006].
Code lines: 212

interface: excursionsetfirstcrossingzhanghuihighorder
Description: Constructors for the Zhang and Hui [2006] excursion set barrier class.
Code lines: 4
Contained by: file structure_formation.excursion_sets.first_crossing_distribution.Zhang_Hui_high_order.F9

function: zhanghuihighorderconstructorinternal
Description: Constructor for the linear barrier excursion set class first crossing class which takes a parameter set as input.
Code lines: 10
Contained by: file structure_formation.excursion_sets.first_crossing_distribution.Zhang_Hui_high_order.F9
Modules used: input_parameters

function: zhanghuihighorderconstructorparameters

Description: Constructor for the linear barrier excursion set class first crossing class which takes a parameter set as input.
Code lines: 10
Contained by: file structure_formation.excursion_sets.first_crossing_distribution.Zhang_Hui_high_order.F9
Modules used: input_parameters

subroutine: zhanghuihighorderinitialize
Description: Initialize the high order Zhang and Hui [2006] excursion set first crossing class.
Code lines: 8
Contained by: file structure_formation.excursion_sets.first_crossing_distribution.Zhang_Hui_high_order.F9

function: zhanghuihighorderprobability
Description: Return the excursion set barrier at the given variance and time.
Code lines: 123
Contained by: file structure_formation.excursion_sets.first_crossing_distribution.Zhang_Hui_high_order.F9
Modules used: galacticus_display iso_c_binding memory_management numerical_interpolation numerical_ranges

file: structure_formation.excursion_sets.first_crossing_distribution.linear_barrier.F90
Description: Contains a module which implements a excursion set first crossing statistics class for linear barriers.
Code lines: 137
Modules used: excursion_sets_barriers

interface: excursionsetfirstcrossinglinearbarrier
Description: Constructors for the linearBarrier excursion set barrier class.
Code lines: 4
Contained by: file structure_formation.excursion_sets.first_crossing_distribution.linear_barrier.F90

function: linearbarrierconstructorinternal
Description: Constructor for the linear barrier excursion set class first crossing class which takes a parameter set as input.
Code lines: 9
Contained by: file structure_formation.excursion_sets.first_crossing_distribution.linear_barrier.F90
Modules used: input_parameters

function: linearbarrierconstructorparameters
Description: Constructor for the linear barrier excursion set class first crossing class which takes a parameter set as input.
Code lines: 13
Contained by: file structure_formation.excursion_sets.first_crossing_distribution.linear_barrier.F90
Modules used: input_parameters

subroutine: linearbarrierdestructor
Description: Destructor for the critical overdensity excursion set barrier class.
Code lines: 7
Contained by: file structure_formation.excursion_sets.first_crossing_distribution.linear_barrier.F90

function: `linearbarrierprobability`
Description: Return the excursion set barrier at the given variance and time.
Code lines: 10
Contained by: file structure_formation.excursion_sets.first_crossing_distribution.linear_barrier.F90
Modules used: numerical_constants_math

function: `linearbarrierrate`
Description: Return the excursion set barrier at the given variance and time.
Code lines: 30
Contained by: file structure_formation.excursion_sets.first_crossing_distribution.linear_barrier.F90
Modules used: numerical_constants_math

function: `barriereffective`
Description: The effective barrier for conditional excursion sets.
Code lines: 7
Contained by: function linearbarrierrate

function: `linearbarrierratenoncrossing`
Description: Return the rate for excursion set non-crossing assuming a linearBarrier barrier. For a linearBarrier barrier the integral over the crossing probability (from zero to infinite variance) equals unity, so all trajectories cross. The non-crossing rate is therefore zero.
Code lines: 12
Contained by: file structure_formation.excursion_sets.first_crossing_distribution.linear_barrier.F90

file: `structure_formation.gravitational_lensing.F90`
Description: Contains a module which implements gravitational lensing from large scale structure.
Code lines: 47

module: `gravitational_lensing`
Description: Implements gravitational lensing from large scale structure.
Code lines: 25
Contained by: file structure_formation.gravitational_lensing.F90
Modules used: iso_c_binding; tables
Used by: subroutine galacticus_function_classes_destroy; file galacticus.output.analyses.distribution_operator.gravitational_lensing.F90; file galacticus.output.analyses.luminosity_function.Halpha.Gunawardhana2013.SDSS.F90; file galacticus.output.analyses.luminosity_function.Halpha.Sobral2013.HiZELS.F90; file galacticus.output.analyses.luminosity_function.Montero-Dorta2009.SDSS.F90; file galacticus.output.analyses.mass_function_HI.ALFALFA_Martin2010.F90; file galacticus.output.analyses.mass_function_stellar.Bernardi_SDSS.F90; file galacticus.output.analyses.mass_function_stellar.GAMA.F90; file galacticus.output.analyses.mass_function_stellar.PRIMUS.F90; file galacticus.output.analyses.mass_function_stellar.SDSS.F90

file galacticus.output.analyses.mass_-function_stellar.UKIDSS_UDS.F90
file galacticus.output.analyses.mass_-function_stellar.ULTRAVISTA.F90
file galacticus.output.analyses.mass_-function_stellar.VIPERS.F90
file galacticus.output.analyses.mass_-function_stellar.ZFOURGE.F90
subroutine galacticus_state_retrieve
subroutine galacticus_state_store

file: structure_formation.gravitational_lensing.Takahashi_2011.F90
Description: Implements the gravitational lensing distributions of Takahashi et al. [2011].
Code lines: 543
Modules used: cosmology_functions, cosmology_parameters, locks, power_spectra_nonlinear

interface: gravitationallensingtakahashi2011
Description: Constructors for the Takahashi et al. [2011] gravitational lensing class.
Code lines: 4
Contained by: file structure_formation.gravitational_lensing.Takahashi_2011.F90

function: takahashi2011constructorinternal
Description: Internal for the Takahashi et al. [2011] gravitational lensing class.
Code lines: 14
Contained by: file structure_formation.gravitational_lensing.Takahashi_2011.F90

function: takahashi2011constructorparameters
Description: Constructor for the Takahashi et al. [2011] gravitational lensing class which takes a parameter list as input.
Code lines: 19
Contained by: file structure_formation.gravitational_lensing.Takahashi_2011.F90
Modules used: input_parameters

function: takahashi2011convergencedistribution
Description: The distribution function for gravitational lensing convergence [Takahashi et al., 2011, eqn. 8].
Code lines: 14
Contained by: file structure_formation.gravitational_lensing.Takahashi_2011.F90

subroutine: takahashi2011destructor
Description: Destructor for the takahasi2011 gravitational lensing class.
Code lines: 9
Contained by: file structure_formation.gravitational_lensing.Takahashi_2011.F90

subroutine: takahashi2011lensingdistributionconstruct
Description: Construct the lensing distribution function for the Takahashi et al. [2011] formalism.
Code lines: 289
Contained by: file structure_formation.gravitational_lensing.Takahashi_2011.F90
Modules used: fgsl, file_utilities, galacticus_error, galacticus_paths, io_hdf5, iso_c_binding, numerical_comparison, numerical_integration, numerical_ranges, root_finder

system_command table_labels

function: convergencedistributionmoment0integrand
Description: Integral over scaled convergence distribution.
Code lines: 7
Contained by: subroutine takahashi2011lensingdistributionconstruct

function: convergencedistributionmoment1integrand
Description: Integral of first moment over scaled convergence distribution.
Code lines: 7
Contained by: subroutine takahashi2011lensingdistributionconstruct

function: convergencedistributionmoment2integrand
Description: Integral of second moment over scaled convergence distribution.
Code lines: 7
Contained by: subroutine takahashi2011lensingdistributionconstruct

function: convergencepdfparametersolver
Description: Root function used in finding equivalent circular orbits.
Code lines: 18
Contained by: subroutine takahashi2011lensingdistributionconstruct

function: convergencevarianceintegrand
Description: Integral for variance in the gravitational lensing convergence.
Code lines: 26
Contained by: subroutine takahashi2011lensingdistributionconstruct
Modules used: numerical_constants_math numerical_constants_physical numerical_constants_prefixes

function: convergencevariancepowerspectrumintegrand
Description: Integral over power spectrum used in computing the variance in the gravitational lensing convergence.
Code lines: 10
Contained by: subroutine takahashi2011lensingdistributionconstruct

function: emptybeamconvergenceintegrand
Description: Integral for gravitational lensing convergence in an empty beam.
Code lines: 15
Contained by: subroutine takahashi2011lensingdistributionconstruct
Modules used: numerical_constants_physical numerical_constants_prefixes

function: magnificationpdfintegrand
Description: Integral for the magnification probability distribution function.
Code lines: 7
Contained by: subroutine takahashi2011lensingdistributionconstruct

function: takahashi2011magnificationcdf
Description: Compute the magnification probability density function at the given `magnification` and `redshift` using the Takahashi et al. [2011] formalism.
Code lines: 52

Contained by: file structure_formation.gravitational_lensing.Takahashi_2011.F90

function: magnificationpdfintegrand
Description: Integral for the magnification probability distribution function.
Code lines: 7
Contained by: function takahashi2011magnificationcdf

function: takahashi2011magnificationdistribution
Description: The gravitational lensing magnification distribution from [Takahashi et al., 2011, eq. 11].
Code lines: 17
Contained by: file structure_formation.gravitational_lensing.Takahashi_2011.F90

function: takahashi2011magnificationpdf
Description: Compute the magnification probability density function at the given **magnification** and **redshift** using the Takahashi et al. [2011] formalism.
Code lines: 34
Contained by: file structure_formation.gravitational_lensing.Takahashi_2011.F90

file: structure_formation.gravitational_lensing.baryonic_modifier.F90
Description: Implements the gravitational lensing distribution by modifying another distribution for the effects of baryons.
Code lines: 194

function: baryonicmodifierconstructorinternal
Description: Generic constructor for the "baryonic modifier" gravitational lensing class.
Code lines: 12
Contained by: file structure_formation.gravitational_lensing.baryonic_modifier.F90

function: baryonicmodifierconstructorparameters
Description: Default constructor for the "baryonic modifier" gravitational lensing class.
Code lines: 32
Contained by: file structure_formation.gravitational_lensing.baryonic_modifier.F90
Modules used: input_parameters

subroutine: baryonicmodifierdestructor
Description: Destructor for the "baryonic modifier" gravitational lensing class.
Code lines: 7
Contained by: file structure_formation.gravitational_lensing.baryonic_modifier.F90

function: baryonicmodifiermagnificationcdf
Description: Compute the magnification probability density function at the given **magnification** and **redshift** by modifying another distribution for the effects of baryons.
Code lines: 16
Contained by: file structure_formation.gravitational_lensing.baryonic_modifier.F90

function: baryonicmodifiermagnificationpdf
Description: Compute the magnification probability density function at the given **magnification** and **redshift** by modifying another distribution for the effects of baryons.
Code lines: 16
Contained by: file structure_formation.gravitational_lensing.baryonic_modifier.F90

subroutine: `baryonicmodifierrenormalize`
Description: Renormlize for PDF for baryonic modification.
Code lines: 48
Contained by: file `structure_formation.gravitational_lensing.baryonic_modifier.F90`
Modules used: `root_finder`

function: `magnificationtransition`
Description: Root finding function used in the "baryonic modifier" gravitational lensing class.
Code lines: 7
Contained by: subroutine `baryonicmodifierrenormalize`

interface: `gravitationallensingbaryonicmodifier`
Description: Constructors for the "baryonic modifier" gravitational lensing class.
Code lines: 4
Contained by: file `structure_formation.gravitational_lensing.baryonic_modifier.F90`

file: `structure_formation.halo_bias.F90`
Description: Contains a module which implements a dark matter halo bias class.
Code lines: 56

module: `dark_matter_halo_biases`
Description: Implements a dark matter halo bias class.
Code lines: 34
Contained by: file `structure_formation.halo_bias.F90`
Modules used: `galacticus_nodes`
Used by:

subroutine `galacticus_function_-classes_destroy`	file `galacticus.output.analyses.correlation_-function.F90`
subroutine `galacticus_state_retrieve`	subroutine `galacticus_state_store`
subroutine `halo_model_projected_-correlation`	file `models.likelihoods.projected_-correlation_function.F90`
file `nodes.property_extractor.halo_-bias.F90`	file `tasks.halo_mass_function.F90`
file `tasks.halo_model.projected_-correlation_function.F90`	file `tasks.mass_function_covariance.F90`

file: `structure_formation.halo_bias.Press-Schechter.F90`
Description: Implementation of halo bias using the Press-Schechter algorithm [Mo and White, 1996].
Code lines: 100
Modules used: `cosmological_density_field`

interface: `darkmatterhalobiaspressschechter`
Description: Constructors for the `pressSchechter` dark matter halo bias class.
Code lines: 4
Contained by: file `structure_formation.halo_bias.Press-Schechter.F90`

function: `pressschechterbiasbymass`
Description: Returns the bias of a dark matter halo given the mass and time.

Code lines: 14
Contained by: file structure_formation.halo_bias.Press-Schechter.F90

function: pressschechterconstructorinternal
Description: Internal constructor for the pressSchechter dark matter halo bias class.
Code lines: 9
Contained by: file structure_formation.halo_bias.Press-Schechter.F90

function: pressschechterconstructorparameters
Description: Constructor for the pressSchechter dark matter halo mass bias which builds the object from a parameter set.
Code lines: 16
Contained by: file structure_formation.halo_bias.Press-Schechter.F90
Modules used: input_parameters

subroutine: pressschechterdestructor
Description: Destructor for the pressSchechter dark matter halo bias class.
Code lines: 8
Contained by: file structure_formation.halo_bias.Press-Schechter.F90

file: structure_formation.halo_bias.Sheth2001.F90
Description: Implementation of halo bias using the algorithm of Sheth et al. [2001].
Code lines: 101
Modules used: cosmological_density_field

interface: darkmatterhalobiassheth2001
Description: Constructors for the sheth2001 dark matter halo bias class.
Code lines: 4
Contained by: file structure_formation.halo_bias.Sheth2001.F90

function: sheth2001biasbymass
Description: Returns the bias of a dark matter halo given the mass and time.
Code lines: 15
Contained by: file structure_formation.halo_bias.Sheth2001.F90

function: sheth2001constructorinternal
Description: Internal constructor for the sheth2001 dark matter halo bias class.
Code lines: 9
Contained by: file structure_formation.halo_bias.Sheth2001.F90

function: sheth2001constructorparameters
Description: Constructor for the sheth2001 dark matter halo mass bias which builds the object from a parameter set.
Code lines: 16
Contained by: file structure_formation.halo_bias.Sheth2001.F90
Modules used: input_parameters

subroutine: sheth2001destructor
Description: Destructor for the sheth2001 dark matter halo bias class.
Code lines: 8

Contained by: file structure_formation.halo_bias.Sheth2001.F90

file: structure_formation.halo_bias.Tinker2010.F90
Description: Implementation of halo bias using the algorithm of Tinker et al. [2010].
Code lines: 123
Modules used: cosmological_density_field virial_density_contrast

interface: darkmatterhalobiastinker2010
Description: Constructors for the tinker2010 dark matter halo bias class.
Code lines: 4
Contained by: file structure_formation.halo_bias.Tinker2010.F90

function: tinker2010biasbymass
Description: Returns the bias of a dark matter halo given the mass and time.
Code lines: 27
Contained by: file structure_formation.halo_bias.Tinker2010.F90

function: tinker2010constructorinternal
Description: Internal constructor for the tinker2010 dark matter halo bias class.
Code lines: 12
Contained by: file structure_formation.halo_bias.Tinker2010.F90

function: tinker2010constructorparameters
Description: Constructor for the tinker2010 dark matter halo mass bias which builds the object from a parameter set.
Code lines: 19
Contained by: file structure_formation.halo_bias.Tinker2010.F90
Modules used: input_parameters

subroutine: tinker2010destructor
Description: Destructor for the tinker2010 dark matter halo bias class.
Code lines: 9
Contained by: file structure_formation.halo_bias.Tinker2010.F90

file: structure_formation.halo_environment.lognormal.F90
Description: Contains a module which implements a log-normal halo environment.
Code lines: 238
Modules used: cosmology_functions cosmology_parameters
linear_growth statistics_distributions

interface: haloenvironmentlognormal
Description: Constructors for the logNormal halo environment class.
Code lines: 4
Contained by: file structure_formation.halo_environment.lognormal.F90

function: lognormalcdf
Description: Return the CDF of the environmental overdensity.
Code lines: 8
Contained by: file structure_formation.halo_environment.lognormal.F90

function: `lognormalconstructorinternal`
Description: Internal constructor for the `logNormal` halo mass function class.
Code lines: 20
Contained by: file `structure_formation.halo_environment.lognormal.F90`
Modules used: `numerical_constants_math`

function: `lognormalconstructorparameters`
Description: Constructor for the `logNormal` halo environment class which takes a parameter set as input.
Code lines: 35
Contained by: file `structure_formation.halo_environment.lognormal.F90`
Modules used: `input_parameters`

subroutine: `lognormaldestructor`
Description: Destructor for the `logNormal` halo mass function class.
Code lines: 11
Contained by: file `structure_formation.halo_environment.lognormal.F90`

function: `lognormalenvironmentmass`
Description: Return the mass of the environment.
Code lines: 8
Contained by: file `structure_formation.halo_environment.lognormal.F90`
Modules used: `numerical_constants_math`

function: `lognormalenvironmentradius`
Description: Return the radius of the environment.
Code lines: 7
Contained by: file `structure_formation.halo_environment.lognormal.F90`

function: `lognormaloverdensitylinear`
Description: Return the environment of the given `node`.
Code lines: 30
Contained by: file `structure_formation.halo_environment.lognormal.F90`
Modules used: `galacticus_nodes` `kind_numbers`

function: `lognormaloverdensitylinearminimum`
Description: Return the minimum overdensity for which the PDF is non-zero.
Code lines: 8
Contained by: file `structure_formation.halo_environment.lognormal.F90`

subroutine: `lognormaloverdensitylinearset`
Description: Return the CDF of the environmental overdensity.
Code lines: 12
Contained by: file `structure_formation.halo_environment.lognormal.F90`
Modules used: `galacticus_error`

function: `lognormaloverdensitynonlinear`
Description: Return the environment of the given `node`.
Code lines: 10

Contained by: file structure_formation.halo_environment.lognormal.F90
Modules used: galacticus_error

function: lognormalpdf
Description: Return the PDF of the environmental overdensity.
Code lines: 8
Contained by: file structure_formation.halo_environment.lognormal.F90

file: structure_formation.halo_environment.normal.F90
Description: Contains a module which implements a normally-distributed halo environment.
Code lines: 282
Modules used: cosmology_functions, cosmology_parameters, kind_numbers, linear_growth, statistics_distributions, tables

interface: haloenvironmentnormal
Description: Constructors for the normal halo environment class.
Code lines: 4
Contained by: file structure_formation.halo_environment.normal.F90

function: normalcdf
Description: Return the CDF of the environmental overdensity.
Code lines: 8
Contained by: file structure_formation.halo_environment.normal.F90

function: normalconstructorinternal
Description: Internal constructor for the normal halo mass function class.
Code lines: 37
Contained by: file structure_formation.halo_environment.normal.F90
Modules used: error_functions, numerical_constants_math

function: normalconstructorparameters
Description: Constructor for the normal halo environment class which takes a parameter set as input.
Code lines: 35
Contained by: file structure_formation.halo_environment.normal.F90
Modules used: input_parameters

subroutine: normaldestructor
Description: Destructor for the normal halo mass function class.
Code lines: 11
Contained by: file structure_formation.halo_environment.normal.F90

function: normalenvironmentmass
Description: Return the mass of the environment.
Code lines: 8
Contained by: file structure_formation.halo_environment.normal.F90
Modules used: numerical_constants_math

function: normalenvironmentradius
Description: Return the radius of the environment.

Code lines: 7
Contained by: file structure_formation.halo_environment.normal.F90

function: normaloverdensitylinear
Description: Return the environment of the given node.
Code lines: 40
Contained by: file structure_formation.halo_environment.normal.F90
Modules used: galacticus_nodes kind_numbers

function: normaloverdensitylinearmaximum
Description: Return the maximum overdensity for which the PDF is non-zero.
Code lines: 7
Contained by: file structure_formation.halo_environment.normal.F90

subroutine: normaloverdensitylinearset
Description: Return the CDF of the environmental overdensity.
Code lines: 14
Contained by: file structure_formation.halo_environment.normal.F90
Modules used: galacticus_error

function: normaloverdensitynonlinear
Description: Return the environment of the given node.
Code lines: 18
Contained by: file structure_formation.halo_environment.normal.F90
Modules used: galacticus_nodes spherical_collapse_matter_lambda

function: normalpdf
Description: Return the PDF of the environmental overdensity.
Code lines: 11
Contained by: file structure_formation.halo_environment.normal.F90

file: structure_formation.halo_environment.uniform.F90
Description: Contains a module which implements a uniform halo environment.
Code lines: 140

interface: haloenvironmentuniform
Description: Constructors for the uniform halo environment class.
Code lines: 3
Contained by: file structure_formation.halo_environment.uniform.F90

function: uniformcdf
Description: Return the CDF of the environmental overdensity.
Code lines: 13
Contained by: file structure_formation.halo_environment.uniform.F90

function: uniformconstructorparameters
Description: Constructor for the uniform halo environment class which takes a parameter set as input.
Code lines: 10
Contained by: file structure_formation.halo_environment.uniform.F90

Modules used: input_parameters

function: uniformenvironmentmass
Description: Return the mass of the environment.
Code lines: 8
Contained by: file structure_formation.halo_environment.uniform.F90

function: uniformenvironmentradius
Description: Return the radius of the environment.
Code lines: 8
Contained by: file structure_formation.halo_environment.uniform.F90

function: uniformoverdensitylinear
Description: Return the environment of the given node.
Code lines: 10
Contained by: file structure_formation.halo_environment.uniform.F90

subroutine: uniformoverdensitylinearset
Description: Return the CDF of the environmental overdensity.
Code lines: 11
Contained by: file structure_formation.halo_environment.uniform.F90
Modules used: galacticus_error

function: uniformoverdensitynonlinear
Description: Return the environment of the given node.
Code lines: 9
Contained by: file structure_formation.halo_environment.uniform.F90

function: uniformpdf
Description: Return the PDF of the environmental overdensity.
Code lines: 11
Contained by: file structure_formation.halo_environment.uniform.F90
Modules used: galacticus_error

file: structure_formation.halo_mass_function.Bhattacharya2011.F90
Description: Contains a module which implements a Bhattacharya et al. [2011] dark matter halo mass function class.
Code lines: 231
Modules used: cosmological_density_field

function: bhattacharya2011a
Description: Return the parameter $\bar{a}$ in the bhattacharya2011 halo mass function at the given time and mass.
Code lines: 9
Contained by: file structure_formation.halo_mass_function.Bhattacharya2011.F90

function: bhattacharya2011constructorinternal
Description: Internal constructor for the bhattacharya2011 halo mass function class.
Code lines: 15
Contained by: file structure_formation.halo_mass_function.Bhattacharya2011.F90

function: `bhattacharya2011constructorparameters`
Description: Constructor for the `bhattacharya2011` halo mass function class which takes a parameter set as input.
Code lines: 57
Contained by: file `structure_formation.halo_mass_function.Bhattacharya2011.F90`
Modules used: `input_parameters`

subroutine: `bhattacharya2011destructor`
Description: Destructor for the `bhattacharya2011` halo mass function class.
Code lines: 9
Contained by: file `structure_formation.halo_mass_function.Bhattacharya2011.F90`

function: `bhattacharya2011differential`
Description: Return the differential halo mass function at the given time and mass.
Code lines: 21
Contained by: file `structure_formation.halo_mass_function.Bhattacharya2011.F90`
Modules used: `numerical_constants_math`

function: `bhattacharya2011normalization`
Description: Return the normalization, $\bar{A}$, in the `bhattacharya2011` halo mass function at the given time and mass.
Code lines: 10
Contained by: file `structure_formation.halo_mass_function.Bhattacharya2011.F90`
Modules used: `numerical_constants_math`

function: `bhattacharya2011p`
Description: Return the parameter $\bar{p}$ in the `bhattacharya2011` halo mass function at the given time and mass.
Code lines: 9
Contained by: file `structure_formation.halo_mass_function.Bhattacharya2011.F90`

function: `bhattacharya2011q`
Description: Return the parameter $\bar{q}$ in the `bhattacharya2011` halo mass function at the given time and mass.
Code lines: 9
Contained by: file `structure_formation.halo_mass_function.Bhattacharya2011.F90`

interface: `halomassfunctionbhattacharya2011`
Description: Constructors for the `bhattacharya2011` halo mass function class.
Code lines: 4
Contained by: file `structure_formation.halo_mass_function.Bhattacharya2011.F90`

file: `structure_formation.halo_mass_function.Despali_2015.F90`
Description: Contains a module which implements a Despali et al. [2015] dark matter halo mass function class.
Code lines: 170
Modules used: `cosmological_density_field` `cosmology_functions` `virial_density_contrast`

function: despali2015a
Description: Return the parameter a in the despali2015 halo mass function at the given time and mass.
Code lines: 11
Contained by: file structure_formation.halo_mass_function.Despali_2015.F90
Modules used: numerical_constants_math

function: despali2015constructorinternal
Description: Internal constructor for the despali2015 halo mass function class.
Code lines: 14
Contained by: file structure_formation.halo_mass_function.Despali_2015.F90

function: despali2015constructorparameters
Description: Constructor for the despali2015 halo mass function class which takes a parameter set as input.
Code lines: 26
Contained by: file structure_formation.halo_mass_function.Despali_2015.F90
Modules used: input_parameters

subroutine: despali2015destructor
Description: Destructor for the despali2015 halo mass function class.
Code lines: 9
Contained by: file structure_formation.halo_mass_function.Despali_2015.F90

function: despali2015normalization
Description: Return the normalization, A, in the despali2015 halo mass function at the given time and mass.
Code lines: 11
Contained by: file structure_formation.halo_mass_function.Despali_2015.F90
Modules used: numerical_constants_math

function: despali2015p
Description: Return the parameter p in the despali2015 halo mass function at the given time and mass.
Code lines: 11
Contained by: file structure_formation.halo_mass_function.Despali_2015.F90
Modules used: numerical_constants_math

function: despali2015x
Description: Return the parameter x in the despali2015 halo mass function at the given time and mass.
Code lines: 9
Contained by: file structure_formation.halo_mass_function.Despali_2015.F90
Modules used: numerical_constants_math

interface: halomassfunctiondespali2015
Description: Constructors for the despali2015 halo mass function class.
Code lines: 4
Contained by: file structure_formation.halo_mass_function.Despali_2015.F90

file: structure_formation.halo_mass_function.F90

Description: Contains a module which provides a class that implements halo mass functions.
Code lines: 163

module: `halo_mass_functions`

Code lines: 141
Contained by: file structure_formation.halo_mass_function.F90
Modules used: cosmology_parameters, galacticus_nodes, fgsl, iso_c_binding
Used by:
- file dark_matter_halos.spins.distributions.N-body_errors.F90
- subroutine galacticus_function_classes_destroy
- file galacticus.output.analyses.spin_distribution.Bett2007.F90
- subroutine galacticus_state_retrieve
- subroutine galacticus_state_store
- subroutine halo_model_projected_correlation
- file merger_trees.construct.build.F90
- file merger_trees.construct.build.masses.distributions.halo_mass_function.F90
- file merger_trees.construct.read.importer.galacticus.F90
- function halomassfunctionevaluate
- file models.likelihoods.mass_function.F90
- file models.likelihoods.projected_correlation_function.F90
- file models.likelihoods.spin_distribution.F90
- file tasks.conditional_mass_function.F90
- file tasks.excursion_sets.F90
- file tasks.halo_mass_function.F90
- file tasks.halo_model.projected_correlation_function.F90
- file tasks.mass_function_covariance.F90
- program tests_halo_mass_function_tinker

function: `integratedintegrand`

Description: Integrand function used to integrate the dark matter halo mass function.
Code lines: 15
Contained by: module halo_mass_functions

function: `massfractionintegrand`

Description: Integrand function used in computing the halo mass fraction.
Code lines: 16
Contained by: module halo_mass_functions

file: `structure_formation.halo_mass_function.Press-Schechter.F90`

Description: Contains a module which implements a Press and Schechter [1974] dark matter halo mass function class.
Code lines: 107
Modules used: cosmological_density_field, excursion_sets_first_crossings

interface: `halomassfunctionpressschechter`

Description: Constructors for the `pressSchechter` halo mass function class.
Code lines: 4
Contained by: file `structure_formation.halo_mass_function.Press-Schechter.F90`

function: `pressschechterconstructorinternal`
Description: Internal constructor for the `pressSchechter` halo mass function class.
Code lines: 10
Contained by: file `structure_formation.halo_mass_function.Press-Schechter.F90`

function: `pressschechterconstructorparameters`
Description: Constructor for the `pressSchechter` halo mass function class which takes a parameter set as input.
Code lines: 19
Contained by: file `structure_formation.halo_mass_function.Press-Schechter.F90`
Modules used: `input_parameters`

subroutine: `pressschechterdestructor`
Description: Destructor for the `pressSchechter` halo mass function class.
Code lines: 9
Contained by: file `structure_formation.halo_mass_function.Press-Schechter.F90`

function: `pressschechterdifferential`
Description: Return the differential halo mass function at the given time and mass.
Code lines: 18
Contained by: file `structure_formation.halo_mass_function.Press-Schechter.F90`
Modules used: `galacticus_error`

file: `structure_formation.halo_mass_function.Rodriguez-Puebla2016.F90`
Description: Contains a module which implements a Tinker et al. [2008] dark matter halo mass function class.
Code lines: 161

interface: `halomassfunctionrodriguezpuebla2016`
Description: Constructors for the `rodriguezPuebla2016` halo mass function class.
Code lines: 4
Contained by: file `structure_formation.halo_mass_function.Rodriguez-Puebla2016.F90`

function: `rodriguezpuebla2016a`
Description: Return the normalization for the `rodriguezPuebla2016` halo mass function class.
Code lines: 13
Contained by: file `structure_formation.halo_mass_function.Rodriguez-Puebla2016.F90`

function: `rodriguezpuebla2016b`
Description: Return the normalization for the `rodriguezPuebla2016` halo mass function class.
Code lines: 13
Contained by: file `structure_formation.halo_mass_function.Rodriguez-Puebla2016.F90`

function: `rodriguezpuebla2016c`
Description: Return the normalization for the `rodriguezPuebla2016` halo mass function class.

Code lines: 13
Contained by: file structure_formation.halo_mass_function.Rodriguez-Puebla2016.F90

function: rodriguezpuebla2016constructorinternal
Description: Internal constructor for the rodriguezPuebla2016 halo mass function class.
Code lines: 20
Contained by: file structure_formation.halo_mass_function.Rodriguez-Puebla2016.F90
Modules used: file_utilities, fox_dom, galacticus_error, galacticus_paths, io_xml, iso_varying_string, table_labels

function: rodriguezpuebla2016constructorparameters
Description: Constructor for the rodriguezPuebla2016 halo mass function class which takes a parameter set as input.
Code lines: 23
Contained by: file structure_formation.halo_mass_function.Rodriguez-Puebla2016.F90
Modules used: input_parameters

subroutine: rodriguezpuebla2016destructor
Description: Destructor for the rodriguezPuebla2016 halo mass function class.
Code lines: 10
Contained by: file structure_formation.halo_mass_function.Rodriguez-Puebla2016.F90

function: rodriguezpuebla2016normalization
Description: Return the normalization for the rodriguezPuebla2016 halo mass function class.
Code lines: 12
Contained by: file structure_formation.halo_mass_function.Rodriguez-Puebla2016.F90

file: structure_formation.halo_mass_function.Sheth-Tormen.F90
Description: Contains a module which implements a Sheth et al. [2001] dark matter halo mass function class.
Code lines: 201
Modules used: cosmological_density_field

interface: halomassfunctionshethtormen
Description: Constructors for the shethTormen halo mass function class.
Code lines: 4
Contained by: file structure_formation.halo_mass_function.Sheth-Tormen.F90

function: shethtormena
Description: Return the parameter a in the shethTormen halo mass function at the given time and mass.
Code lines: 10
Contained by: file structure_formation.halo_mass_function.Sheth-Tormen.F90
Modules used: numerical_constants_math

function: shethtormenconstructorinternal
Description: Internal constructor for the shethTormen halo mass function class.
Code lines: 14
Contained by: file structure_formation.halo_mass_function.Sheth-Tormen.F90

function: `shethtormenconstructorparameters`
Description: Constructor for the `shethTormen` halo mass function class which takes a parameter set as input.
Code lines: 45
Contained by: file `structure_formation.halo_mass_function.Sheth-Tormen.F90`
Modules used: `input_parameters`

subroutine: `shethtormendestructor`
Description: Destructor for the `shethTormen` halo mass function class.
Code lines: 9
Contained by: file `structure_formation.halo_mass_function.Sheth-Tormen.F90`

function: `shethtormendifferential`
Description: Return the differential halo mass function at the given time and mass.
Code lines: 21
Contained by: file `structure_formation.halo_mass_function.Sheth-Tormen.F90`
Modules used: `numerical_constants_math`

function: `shethtormennormalization`
Description: Return the normalization, A, in the `shethTormen` halo mass function at the given time and mass.
Code lines: 10
Contained by: file `structure_formation.halo_mass_function.Sheth-Tormen.F90`
Modules used: `numerical_constants_math`

function: `shethtormenp`
Description: Return the parameter p in the `shethTormen` halo mass function at the given time and mass.
Code lines: 10
Contained by: file `structure_formation.halo_mass_function.Sheth-Tormen.F90`
Modules used: `numerical_constants_math`

file: `structure_formation.halo_mass_function.Tinker2008.F90`
Description: Contains a module which implements a Tinker et al. [2008] dark matter halo mass function class.
Code lines: 237
Modules used: `tables` `virial_density_contrast`

interface: `halomassfunctiontinker2008`
Description: Constructors for the `tinker2008` halo mass function class.
Code lines: 4
Contained by: file `structure_formation.halo_mass_function.Tinker2008.F90`

function: `tinker2008a`
Description: Return the normalization for the `tinker2008` halo mass function class.
Code lines: 11
Contained by: file `structure_formation.halo_mass_function.Tinker2008.F90`

function: `tinker2008b`
Description: Return the normalization for the `tinker2008` halo mass function class.

Code lines: 11
Contained by: file structure_formation.halo_mass_function.Tinker2008.F90

function: tinker2008c
Description: Return the normalization for the tinker2008 halo mass function class.
Code lines: 11
Contained by: file structure_formation.halo_mass_function.Tinker2008.F90

function: tinker2008constructorinternal
Description: Internal constructor for the tinker2008 halo mass function class.
Code lines: 46
Contained by: file structure_formation.halo_mass_function.Tinker2008.F90
Modules used: file_utilities, fox_dom, galacticus_error, galacticus_paths, io_xml, iso_varying_string, table_labels

function: tinker2008constructorparameters
Description: Constructor for the tinker2008 halo mass function class which takes a parameter set as input.
Code lines: 26
Contained by: file structure_formation.halo_mass_function.Tinker2008.F90
Modules used: input_parameters

subroutine: tinker2008destructor
Description: Destructor for the tinker2008 halo mass function class.
Code lines: 12
Contained by: file structure_formation.halo_mass_function.Tinker2008.F90

function: tinker2008normalization
Description: Return the normalization for the tinker2008 halo mass function class.
Code lines: 11
Contained by: file structure_formation.halo_mass_function.Tinker2008.F90

subroutine: tinker2008parametersevaluate
Description: Evaluate interpolating parametersn for the tinker2008 halo mass function class.
Code lines: 20
Contained by: file structure_formation.halo_mass_function.Tinker2008.F90

file: structure_formation.halo_mass_function.Tinker2008Form.F90
Description: Contains a module which implements a Tinker et al. [2008] dark matter halo mass function class.
Code lines: 105
Modules used: cosmological_density_field, cosmology_functions, linear_growth

type: halomassfunctiontinker2008form
Description: A halo mass function class using the fitting function of Tinker et al. [2008].
Code lines: 40
Contained by: file structure_formation.halo_mass_function.Tinker2008Form.F90

function: `tinker2008formdifferential`
Description: Return the differential halo mass function at the given time and mass.
Code lines: 24
Contained by: file `structure_formation.halo_mass_function.Tinker2008Form.F90`
Modules used: `numerical_interpolation` `table_labels`

file: `structure_formation.halo_mass_function.Tinker2008Generic.F90`
Description: Contains a module which implements a Tinker et al. [2008] dark matter halo mass function class with user-specified parameters.
Code lines: 181

interface: `halomassfunctiontinker2008generic`
Description: Constructors for the `tinker2008Generic` halo mass function class.
Code lines: 4
Contained by: file `structure_formation.halo_mass_function.Tinker2008Generic.F90`

function: `tinker2008generica`
Description: Return the a parameter for the `tinker2008Generic` halo mass function class.
Code lines: 9
Contained by: file `structure_formation.halo_mass_function.Tinker2008Generic.F90`

function: `tinker2008genericb`
Description: Return the b parameter for the `tinker2008Generic` halo mass function class.
Code lines: 9
Contained by: file `structure_formation.halo_mass_function.Tinker2008Generic.F90`

function: `tinker2008genericc`
Description: Return the c parameter for the `tinker2008Generic` halo mass function class.
Code lines: 9
Contained by: file `structure_formation.halo_mass_function.Tinker2008Generic.F90`

function: `tinker2008genericconstructorinternal`
Description: Internal constructor for the `tinker2008Generic` halo mass function class.
Code lines: 18
Contained by: file `structure_formation.halo_mass_function.Tinker2008Generic.F90`

function: `tinker2008genericconstructorparameters`
Description: Constructor for the `tinker2008Generic` halo mass function class which takes a parameter set as input.
Code lines: 59
Contained by: file `structure_formation.halo_mass_function.Tinker2008Generic.F90`
Modules used: `input_parameters`

subroutine: `tinker2008genericdestructor`
Description: Destructor for the `tinker2008Generic` halo mass function class.
Code lines: 10
Contained by: file `structure_formation.halo_mass_function.Tinker2008Generic.F90`

function: `tinker2008genericnormalization`
Description: Return the normalization for the `tinker2008Generic` halo mass function class.
Code lines: 9
Contained by: file structure_formation.halo_mass_function.Tinker2008Generic.F90

file: `structure_formation.halo_mass_function.environment_averaged.F90`
Description: Contains a module which implements a dark matter halo mass function class which averages another (presumably environment-dependent) mass function over environment.
Code lines: 182
Modules used: cosmological_density_field

function: `environmentaveragedconstructorinternal`
Description: Internal constructor for the `environmentAveraged` halo mass function class.
Code lines: 11
Contained by: file structure_formation.halo_mass_function.environment_averaged.F90

function: `environmentaveragedconstructorparameters`
Description: Constructor for the `environmentAveraged` halo mass function class which takes a parameter set as input.
Code lines: 22
Contained by: file structure_formation.halo_mass_function.environment_averaged.F90
Modules used: input_parameters

subroutine: `environmentaverageddestructor`
Description: Destructor for the `environmentAveraged` halo mass function class.
Code lines: 10
Contained by: file structure_formation.halo_mass_function.environment_averaged.F90

function: `environmentaverageddifferential`
Description: Return the differential halo mass function at the given time and mass.
Code lines: 84
Contained by: file structure_formation.halo_mass_function.environment_averaged.F90
Modules used: galacticus_nodes, iso_c_binding, numerical_integration, root_finder

function: `environmentaveragedintegrand`
Description: Integrand function used in averging the dark matter halo mass function over environment.
Code lines: 8
Contained by: function environmentaverageddifferential

function: `environmentaveragedroot`
Description: Root function used in determining the range of overdensity over which to integrate.
Code lines: 7
Contained by: function environmentaverageddifferential

interface: `halomassfunctionenvironmentaveraged`
Description: Constructors for the `environmentAveraged` halo mass function class.
Code lines: 4

Contained by: file structure_formation.halo_mass_function.environment_averaged.F90

file: structure_formation.halo_mass_function.environmental.F90
Description: Contains a module which implements a dark matter halo mass function class which handles the transition through the environment mass scale.
Code lines: 100
Modules used: cosmological_density_field

function: environmentalconstructorinternal
Description: Internal constructor for the environmental halo mass function class.
Code lines: 11
Contained by: file structure_formation.halo_mass_function.environmental.F90

function: environmentalconstructorparameters
Description: Constructor for the environmental halo mass function class which takes a parameter set as input.
Code lines: 22
Contained by: file structure_formation.halo_mass_function.environmental.F90
Modules used: input_parameters

function: environmentaldifferential
Description: Return the differential halo mass function at the given time and mass.
Code lines: 17
Contained by: file structure_formation.halo_mass_function.environmental.F90

interface: halomassfunctionenvironmental
Description: Constructors for the environmental halo mass function class.
Code lines: 4
Contained by: file structure_formation.halo_mass_function.environmental.F90

file: structure_formation.halo_mass_function.error_convolved.F90
Description: Contains a module which implements a dark matter halo mass function class which modifies another mass function by convolving with a mass-dependent error.
Code lines: 163
Modules used: statistics_nbody_halo_mass_errors

function: errorconvolvedconstructorinternal
Description: Internal constructor for the errorConvolved halo mass function class.
Code lines: 11
Contained by: file structure_formation.halo_mass_function.error_convolved.F90

function: errorconvolvedconstructorparameters
Description: Constructor for the errorConvolved halo mass function class which takes a parameter set as input.
Code lines: 28
Contained by: file structure_formation.halo_mass_function.error_convolved.F90
Modules used: input_parameters

subroutine: errorconvolveddestructor
Description: Destructor for the errorConvolved halo mass function class.

Code lines: 9
Contained by: file structure_formation.halo_mass_function.error_convolved.F90

function: errorconvolveddifferential
Description: Return the differential halo mass function at the given time and mass.
Code lines: 53
Contained by: file structure_formation.halo_mass_function.error_convolved.F90
Modules used: galacticus_nodes iso_c_binding numerical_integration

function: errorconvolvedconvolution
Description: Integrand function used in convolving the dark matter halo mass function.
Code lines: 12
Contained by: function errorconvolveddifferential
Modules used: numerical_constants_math

interface: halomassfunctionerrorconvolved
Description: Constructors for the errorConvolved halo mass function class.
Code lines: 4
Contained by: file structure_formation.halo_mass_function.error_convolved.F90

file: structure_formation.halo_mass_function.friends_of_friends_bias.F90
Description: Contains a module which implements a dark matter halo mass function class which modifies another mass function to mimic systematic errors arising in the friends-of-friends halo finding algorithm.
Code lines: 229
Modules used: dark_matter_halo_scales dark_matter_profiles_dmo

function: fofbiasconstructorinternal
Description: Internal constructor for the fofBias halo mass function class.
Code lines: 14
Contained by: file structure_formation.halo_mass_function.friends_of_friends_bias.F90

function: fofbiasconstructorparameters
Description: Constructor for the fofBias halo mass function class which takes a parameter set as input.
Code lines: 57
Contained by: file structure_formation.halo_mass_function.friends_of_friends_bias.F90
Modules used: input_parameters

subroutine: fofbiasdestructor
Description: Destructor for the fofBias halo mass function class.
Code lines: 11
Contained by: file structure_formation.halo_mass_function.friends_of_friends_bias.F90

function: fofbiasdifferential
Description: Return the differential halo mass function at the given time and mass.
Code lines: 83
Contained by: file structure_formation.halo_mass_function.friends_of_friends_bias.F90
Modules used: galacticus_error galacticus_nodes numerical_constants_math

interface: `halomassfunctionfofbias`
Description: Constructors for the `fofBias` halo mass function class.
Code lines: 4
Contained by: file structure_formation.halo_mass_function.friends_of_friends_bias.F90

file: `structure_formation.halo_mass_function.simple_systematic.F90`
Description: Contains a module which implements a dark matter halo mass function class which modifies another mass function using a simple model for systematics.
Code lines: 112

interface: `halomassfunctionsimplesystematic`
Description: Constructors for the `simpleSystematic` halo mass function class.
Code lines: 4
Contained by: file structure_formation.halo_mass_function.simple_systematic.F90

function: `simplesystematicconstructorinternal`
Description: Internal constructor for the `simpleSystematic` halo mass function class.
Code lines: 10
Contained by: file structure_formation.halo_mass_function.simple_systematic.F90

function: `simplesystematicconstructorparameters`
Description: Constructor for the `simpleSystematic` halo mass function class which takes a parameter set as input.
Code lines: 34
Contained by: file structure_formation.halo_mass_function.simple_systematic.F90
Modules used: input_parameters

subroutine: `simplesystematicdestructor`
Description: Destructor for the `simpleSystematic` halo mass function class.
Code lines: 8
Contained by: file structure_formation.halo_mass_function.simple_systematic.F90

function: `simplesystematicdifferential`
Description: Return the differential halo mass function at the given time and mass.
Code lines: 11
Contained by: file structure_formation.halo_mass_function.simple_systematic.F90

file: `structure_formation.linear_growth.F90`
Description: Contains a module which provides a class that implements linear growth of cosmological structure.
Code lines: 68

module: `linear_growth`
Description: Provides an object that implements linear growth of cosmological structure.
Code lines: 46
Contained by: file structure_formation.linear_growth.F90
Used by: subroutine dark_matter_halo_-correa2015_fit_parameters file dark_matter_halos.mass_accretion_-history.Correa2015.F90

file dark_matter_profiles.structure.concentration.Correa2015.F90	file dark_matter_profiles.structure.concentration.Prada2011.F90
subroutine galacticus_function_classes_destroy	file galacticus.output.analyses.correlation_function.F90
subroutine galacticus_state_retrieve	subroutine galacticus_state_store
subroutine halo_model_projected_correlation	file intergalactic_medium.filtering_mass.Gnedin2000.F90
file models.likelihoods.projected_correlation_function.F90	file structure_formation.critical_overdensity.Kitayama_Suto1996.F90
file structure_formation.critical_overdensity.environmental.F90	file structure_formation.critical_overdensity.fixed.F90
file structure_formation.critical_overdensity.spherical_collapse_matter_lambda.F90	function sphericalcollapsematterlambdagradientmass
file structure_formation.halo_environment.lognormal.F90	file structure_formation.halo_environment.normal.F90
file structure_formation.halo_mass_function.Tinker2008Form.F90	file structure_formation.power_spectrum.nonlinear.PeacockDodds1996.F90
function peacockdodds1996value	file structure_formation.power_spectrum.nonlinear.linear.F90
subroutine make_table	subroutine spherical_collapse_dark_energy_critical_overdensity_tabulate
subroutine make_table	subroutine spherical_collapse_matter_lambda_critical_overdensity_tabulate
subroutine spherical_collapse_matter_lambda_nonlinear_mapping	file tasks.halo_mass_function.F90
file tasks.halo_model.projected_correlation_function.F90	file tasks.power_spectrum.F90
program tests_linear_growth_eds	program tests_linear_growth_cosmological_constant
program tests_linear_growth_dark_energy	program tests_linear_growth_open
program tests_spherical_collapse_dark_energy_omega_half	program tests_spherical_collapse_dark_energy_omega_two_thirds
program tests_spherical_collapse_dark_energy_lambda	program tests_spherical_collapse_dark_energy_open
program tests_spherical_collapse_nonlinear	file universe.operators.intergalactic_medium_state_evolve.F90

file: `structure_formation.linear_growth.simple.F90`

Description: An implementation of linear growth of cosmological structure in simple cosomologies. Ignores pressure terms for the growth of baryons and has no wavenumber dependence. Also assumes no growth of radiation perturbations.

Code lines: 259

Modules used: cosmology_functions cosmology_parameters tables

interface: `lineargrowthsimple`

Description: Constructors for the `simple` linear growth class.
Code lines: 4
Contained by: file structure_formation.linear_growth.simple.F90

function: `simpleconstructorinternal`
Description: Internal constructor for the `simple` linear growth class.
Code lines: 19
Contained by: file structure_formation.linear_growth.simple.F90

function: `simpleconstructorparameters`
Description: Constructor for the `simple` linear growth class which takes a parameter set as input.
Code lines: 16
Contained by: file structure_formation.linear_growth.simple.F90
Modules used: input_parameters

subroutine: `simpledestructor`
Description: Destructor for the `simple` linear growth class.
Code lines: 12
Contained by: file structure_formation.linear_growth.simple.F90

function: `simplelogarithmicderivativeexpansionfactor`
Description: Return the logarithmic gradient of linear growth factor with respect to expansion factor at the given epoch.
Code lines: 19
Contained by: file structure_formation.linear_growth.simple.F90
Modules used: galacticus_error

subroutine: `simpleretabulate`
Description: Returns the linear growth factor $D(a)$ for expansion factor `aExpansion`, normalized such that $D(1) = 1$ for a simple matter plus cosmological constant cosmology.
Code lines: 94
Contained by: file structure_formation.linear_growth.simple.F90
Modules used: `fgsl` ode_solver table_labels tables

function: `growthfactorodes`
Description: System of differential equations to solve for the growth factor.
Code lines: 11
Contained by: subroutine simpleretabulate

function: `simplevalue`
Description: Return the linear growth factor at the given epoch.
Code lines: 24
Contained by: file structure_formation.linear_growth.simple.F90

file: `structure_formation.power_spectrum.F90`
Description: Contains a module which implements a cosmological power spectra class.
Code lines: 51

module: `power_spectra`

Description: Implements linear-theory power spectra.
Code lines: 29
Contained by: file structure_formation.power_spectrum.F90
Used by:
- file dark_matter_profiles.structure.concentration.Diemer-Kravtsov_2014.F90
- subroutine galacticus_function_classes_201[illegible]
- file galacticus.output.analyses.correlation_function.F90
- subroutine galacticus_state_retrieve
- subroutine galacticus_state_store
- subroutine halo_model_projected_correlation
- file models.likelihoods.projected_correlation_function.F90
- file structure_formation.power_spectrum.nonlinear.PeacockDodds1996.F90
- file structure_formation.power_spectrum.nonlinear.linear.F90
- file tasks.excursion_sets.F90
- file tasks.halo_model.projected_correlation_function.F90
- file tasks.power_spectrum.F90
- program test_diemerkravtsov2014_concentration
- program tests_power_spectrum

file: `structure_formation.power_spectrum.nonlinear.CosmicEmu.F90`

Description: Contains a module which implements a nonlinear power spectrum class in which the nonlinear power spectrum is computed using the code of Lawrence et al. [2010].
Code lines: 245
Modules used:
- cosmological_density_field
- cosmology_functions
- cosmology_parameters
- fgsl
- file_utilities
- power_spectra_primordial

function: `cosmicemuconstructorinternal`

Description: Internal constructor for the `CosmicEmu` nonlinear power spectrum class.
Code lines: 21
Contained by: file structure_formation.power_spectrum.nonlinear.CosmicEmu.F90
Modules used:
- galacticus_error
- numerical_comparison

function: `cosmicemuconstructorparameters`

Description: Constructor for the cosmicEmu nonlinear power spectrum class which takes a parameter set as input.
Code lines: 25
Contained by: file structure_formation.power_spectrum.nonlinear.CosmicEmu.F90
Modules used: input_parameters

subroutine: `cosmicemudestructor`

Description: Destructor for the `CosmicEmu` nonlinear power spectrum class.
Code lines: 10
Contained by: file structure_formation.power_spectrum.nonlinear.CosmicEmu.F90

function: `cosmicemuvalue`

Description: Return a nonlinear power spectrum equal using the code of Lawrence et al. [2010].

Code lines: 120
Contained by: file structure_formation.power_spectrum.nonlinear.CosmicEmu.F90
Modules used: galacticus_display, galacticus_error, galacticus_paths, iso_varying_string, memory_management, numerical_comparison, numerical_interpolation, system_command, table_labels

interface: powerspectrumnonlinearcosmicemu
Description: Constructors for the CosmicEmu nonlinear power spectrum class.
Code lines: 4
Contained by: file structure_formation.power_spectrum.nonlinear.CosmicEmu.F90

file: structure_formation.power_spectrum.nonlinear.F90
Description: Contains a module which implements the nonlinear power spectrum.
Code lines: 39

module: power_spectra_nonlinear
Description: Implements the nonlinear power spectrum.
Code lines: 17
Contained by: file structure_formation.power_spectrum.nonlinear.F90
Used by: subroutine galacticus_function_classes_destroy, subroutine galacticus_state_retrieve, subroutine galacticus_state_store, file structure_formation.gravitational_lensing.Takahashi_2011.F90, file tasks.mass_function_covariance.F90, file tasks.power_spectrum.F90

file: structure_formation.power_spectrum.nonlinear.PeacockDodds1996.F90
Description: Contains a module which implements a nonlinear power spectrum class in which the nonlinear power spectrum is computed using the algorithm of Peacock and Dodds [1996].
Code lines: 174
Modules used: cosmology_functions, linear_growth, power_spectra

function: peacockdodds1996constructorinternal
Description: Internal constructor for the PeacockDodds1996 nonlinear power spectrum class.
Code lines: 13
Contained by: file structure_formation.power_spectrum.nonlinear.PeacockDodds1996.F90

function: peacockdodds1996constructorparameters
Description: Constructor for the peacockDodds1996 nonlinear power spectrum class which takes a parameter set as input.
Code lines: 22
Contained by: file structure_formation.power_spectrum.nonlinear.PeacockDodds1996.F90
Modules used: input_parameters

subroutine: peacockdodds1996destructor
Description: Destructor for the PeacockDodds1996 nonlinear power spectrum class.

Code lines: 9
Contained by: file structure_formation.power_spectrum.nonlinear.PeacockDodds1996.F90

function: peacockdodds1996value
Description: Return a nonlinear power spectrum equal using the algorithm of Peacock and Dodds [1996].
Code lines: 73
Contained by: file structure_formation.power_spectrum.nonlinear.PeacockDodds1996.F90
Modules used: cosmology_functions galacticus_error linear_growth numerical_constants_math

interface: powerspectrumnonlinearpeacockdodds1996
Description: Constructors for the PeacockDodds1996 nonlinear power spectrum class.
Code lines: 4
Contained by: file structure_formation.power_spectrum.nonlinear.PeacockDodds1996.F90

file: structure_formation.power_spectrum.nonlinear.linear.F90
Description: Contains a module which implements a nonlinear power spectrum class in which the nonlinear power spectrum is just the linear power spectrum. Intended primarily for testing purposes.
Code lines: 94
Modules used: linear_growth power_spectra

function: linearconstructorinternal
Description: Internal constructor for the linear nonlinear power spectrum class.
Code lines: 9
Contained by: file structure_formation.power_spectrum.nonlinear.linear.F90

function: linearconstructorparameters
Description: Constructor for the linear nonlinear power spectrum class which takes a parameter set as input.
Code lines: 16
Contained by: file structure_formation.power_spectrum.nonlinear.linear.F90
Modules used: input_parameters

subroutine: lineardestructor
Description: Destructor for the linear nonlinear power spectrum class.
Code lines: 8
Contained by: file structure_formation.power_spectrum.nonlinear.linear.F90

function: linearvalue
Description: Return the nonlinear power spectrum at the given wavenumber.
Code lines: 8
Contained by: file structure_formation.power_spectrum.nonlinear.linear.F90

interface: powerspectrumnonlinearlinear
Description: Constructors for the linear nonlinear power spectrum class.
Code lines: 4
Contained by: file structure_formation.power_spectrum.nonlinear.linear.F90

file: `structure_formation.power_spectrum.primordial.F90`
Description: Contains a module which provides a class that implements the primordial power spectrum.
Code lines: 45

module: `power_spectra_primordial`
Description: Provides a class that implements the primordial power spectrum.
Code lines: 23
Contained by: file structure_formation.power_spectrum.primordial.F90
Used by: subroutine galacticus_function_classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; file structure_formation.power_spectrum.nonlinear.CosmicEmu.F90; file structure_formation.power_spectrum.primordial.transferred.simple.F90; file tasks.merger_tree_file_builder.F90; program tests_power_spectrum; program tests_sigma

file: `structure_formation.power_spectrum.primordial.power_law.F90`
Description: A primordial power spectrum class which provides a power-law power spectrum.
Code lines: 128

function: `powerlawconstructorinternal`
Description: Internal constructor for the "power-law" primordial power spectrum class.
Code lines: 10
Contained by: file structure_formation.power_spectrum.primordial.power_law.F90

function: `powerlawconstructorparameters`
Description: Constructor for the "power-law" primordial power spectrum class which takes a parameter set as input.
Code lines: 37
Contained by: file structure_formation.power_spectrum.primordial.power_law.F90
Modules used: galacticus_display

subroutine: `powerlawdestructor`
Description: Destructor for the "power-law" primordial power spectrum class.
Code lines: 8
Contained by: file structure_formation.power_spectrum.primordial.power_law.F90

function: `powerlawlogarithmicderivative`
Description: Return the logarithmic derivative of the primordial power spectrum at the given `wavenumber`.
Code lines: 8
Contained by: file structure_formation.power_spectrum.primordial.power_law.F90

function: `powerlawpower`
Description: Return the primordial power spectrum at the given `wavenumber`.
Code lines: 8
Contained by: file structure_formation.power_spectrum.primordial.power_law.F90

interface: `powerspectrumprimordialpowerlaw`
Description: Constructors for the "power-law" primordial power spectrum class.
Code lines: 4
Contained by: file `structure_formation.power_spectrum.primordial.power_law.F90`

file: `structure_formation.power_spectrum.primordial.transferred.F90`
Description: Contains a module which provides a class that implements the transferred primordial power spectrum.
Code lines: 45

module: `power_spectra_primordial_transferred`
Description: Provides a class that implements the transferred primordial power spectrum.
Code lines: 23
Contained by: file `structure_formation.power_spectrum.primordial.transferred.F90`
Used by:
- subroutine `galacticus_function_classes_destroy`
- subroutine `galacticus_state_retrieve`
- subroutine `galacticus_state_store`
- file `structure_formation.cosmological_mass_variance.filtered_power_spectrum.F90`
- file `structure_formation.power_spectrum.standard.F90`
- program `tests_power_spectrum`
- program `tests_sigma`

file: `structure_formation.power_spectrum.primordial.transferred.simple.F90`
Description: A simple transferred primordial power spectrum class.
Code lines: 107
Modules used: `power_spectra_primordial`, `transfer_functions`

interface: `powerspectrumprimordialtransferredsimple`
Description: Constructors for the "simple" transferred primordial power spectrum class.
Code lines: 4
Contained by: file `structure_formation.power_spectrum.primordial.transferred.simple.F90`

function: `simpleconstructorinternal`
Description: Internal constructor for the "simple" transferred primordial power spectrum class.
Code lines: 9
Contained by: file `structure_formation.power_spectrum.primordial.transferred.simple.F90`

function: `simpleconstructorparameters`
Description: Constructor for the "simple" transferred primordial power spectrum class which takes a parameter set as input.
Code lines: 17
Contained by: file `structure_formation.power_spectrum.primordial.transferred.simple.F90`
Modules used: `input_parameters`

subroutine: `simpledestructor`
Description: Destructor for the "simple" transferred primordial power spectrum class.
Code lines: 8

Contained by: file structure_formation.power_spectrum.primordial.transferred.simple.F90

function: simplelogarithmicderivative
Description: Return the logarithmic derivative of the transferred primordial power spectrum at the given wavenumber.
Code lines: 9
Contained by: file structure_formation.power_spectrum.primordial.transferred.simple.F90

function: simplepower
Description: Return the transferred primordial power spectrum at the given wavenumber.
Code lines: 9
Contained by: file structure_formation.power_spectrum.primordial.transferred.simple.F90

file: structure_formation.power_spectrum.standard.F90
Description: Contains a module which implements a linear theory power spectrum class in which the power spectrum is just the transferred primordial power spectrum correctly normalized to $z = 0$.
Code lines: 120
Modules used: cosmological_density_field power_spectra_primordial_transferred

interface: powerspectrumstandard
Description: Constructors for the standard power spectrum class.
Code lines: 4
Contained by: file structure_formation.power_spectrum.standard.F90

function: standardconstructorinternal
Description: Internal constructor for the standard nonstandard power spectrum class.
Code lines: 9
Contained by: file structure_formation.power_spectrum.standard.F90

function: standardconstructorparameters
Description: Constructor for the standard nonstandard power spectrum class which takes a parameter set as input.
Code lines: 16
Contained by: file structure_formation.power_spectrum.standard.F90
Modules used: input_parameters

subroutine: standarddestructor
Description: Destructor for the standard nonstandard power spectrum class.
Code lines: 8
Contained by: file structure_formation.power_spectrum.standard.F90

function: standardpower
Description: Return the cosmological power spectrum for $k =$wavenumber [Mpc^{-1}].
Code lines: 9
Contained by: file structure_formation.power_spectrum.standard.F90

function: standardpowerdimensionless
Description: Return the dimensionless power spectrum, $\Delta^2(k)$, for $k =$wavenumber [Mpc^{-1}].
Code lines: 9

Contained by: file structure_formation.power_spectrum.standard.F90
Modules used: numerical_constants_math

function: standardpowerlogarithmicderivative
Description: Return the logarithmic derivative of the power spectrum, $\mathrm{d}\ln P(k)/\mathrm{d}\ln k$, for $k =$wavenumber [Mpc^{-1}].
Code lines: 9
Contained by: file structure_formation.power_spectrum.standard.F90

file: structure_formation.power_spectrum.variance.window_function.F90
Description: Contains a module which provides a class that implements window functions for computing the variance of the power spectrum.
Code lines: 57

module: power_spectrum_window_functions
Description: Provides a class which implements window functions for computing the variance of the power spectrum.
Code lines: 34
Contained by: file structure_formation.power_spectrum.variance.window_function.F90
Used by: subroutine galacticus_function_classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; file structure_formation.cosmological_mass_variance.filtered_power_spectrum.F90; file tasks.power_spectrum.F90; program tests_sigma

file: structure_formation.power_spectrum.variance.window_function.Lagrangian_Chan2017.F90
Description: Contains a module which provides a power spectrum window function class that implements the Lagrangian filter of Chan et al. [2017].
Code lines: 133
Modules used: cosmology_parameters

function: lagrangianchan2017constructorinternal
Description: Internal constructor for the lagrangianChan2017 power spectrum window function class.
Code lines: 11
Contained by: file structure_formation.power_spectrum.variance.window_function.Lagrangian_Chan2017.F90
Modules used: numerical_constants_math

function: lagrangianchan2017constructorparameters
Description: Constructor for the lagrangianChan2017 power spectrum window function class which takes a parameter set as input.
Code lines: 22
Contained by: file structure_formation.power_spectrum.variance.window_function.Lagrangian_Chan2017.F90
Modules used: input_parameters

subroutine: lagrangianchan2017destructor
Description: Destructor for the lagrangianChan2017 power spectrum window function class.
Code lines: 7
Contained by: file structure_formation.power_spectrum.variance.window_function.Lagrangian_Chan2017.F90

function: `lagrangianchan2017value`
Description: Smooth-k space power spectrum window function proposed in Leo et al. [2018]. spectrum.
Code lines: 27
Contained by: file `structure_formation.power_spectrum.variance.window_function.Lagrangian_Chan2017.F90`
Modules used: `cosmology_parameters` `numerical_constants_math`

function: `lagrangianchan2017wavenumbermaximum`
Description: Maximum wavenumber for a top hat in real space window function Fourier transformed into k-space used in computing the variance of the power spectrum.
Code lines: 11
Contained by: file `structure_formation.power_spectrum.variance.window_function.Lagrangian_Chan2017.F90`

interface: `powerspectrumwindowfunctionlagrangianchan2017`
Description: Constructors for the `lagrangianChan2017` power spectrum window function class.
Code lines: 4
Contained by: file `structure_formation.power_spectrum.variance.window_function.Lagrangian_Chan2017.F90`

file: `structure_formation.power_spectrum.variance.window_function.sharp_kSpace.F90`
Description: Contains a module which implements a sharp k-space power spectrum window function class.
Code lines: 156
Modules used: `cosmology_parameters`

interface: `powerspectrumwindowfunctionsharpkspace`
Description: Constructors for the `sharpKSpace` power spectrum window function class.
Code lines: 4
Contained by: file `structure_formation.power_spectrum.variance.window_function.sharp_kSpace.F90`

function: `sharpkspaceamplitudeismassindependent`
Description: Indicate the the sharp k-space power spectrum window function has constant amplitude below the maximum wavenumber.
Code lines: 8
Contained by: file `structure_formation.power_spectrum.variance.window_function.sharp_kSpace.F90`

function: `sharpkspaceconstructorinternal`
Description: Internal constructor for the `sharpKSpace` power spectrum window function class.
Code lines: 22
Contained by: file `structure_formation.power_spectrum.variance.window_function.sharp_kSpace.F90`
Modules used: `numerical_constants_math`

function: `sharpkspaceconstructorparameters`
Description: Constructor for the `sharpKSpace` power spectrum window function class which takes a parameter set as input.
Code lines: 37
Contained by: file `structure_formation.power_spectrum.variance.window_function.sharp_kSpace.F90`
Modules used: `input_parameters`

subroutine: `sharpkspacedestructor`

Description: Destructor for the `sharpKSpace` power spectrum window function class.
Code lines: 7
Contained by: file `structure_formation.power_spectrum.variance.window_function.sharp_kSpace.F90`

function: `sharpkspacevalue`
Description: Sharp k-space window function used in computing the variance of the power spectrum.
Code lines: 16
Contained by: file `structure_formation.power_spectrum.variance.window_function.sharp_kSpace.F90`

function: `sharpkspacewavenumbermaximum`
Description: Sharp k-space window function used in computing the variance of the power spectrum.
Code lines: 8
Contained by: file `structure_formation.power_spectrum.variance.window_function.sharp_kSpace.F90`

file: `structure_formation.power_spectrum.variance.window_function.smooth_k_space.F90`
Description: Contains a module which provides a power spectrum window function class that implements the smooth-k space filter of Leo et al. [2018].
Code lines: 138
Modules used: `cosmology_parameters`

interface: `powerspectrumwindowfunctionsmoothkspace`
Description: Constructors for the `smoothKSpace` power spectrum window function class.
Code lines: 4
Contained by: file `structure_formation.power_spectrum.variance.window_function.smooth_k_space.F90`

function: `smoothkspaceconstructorinternal`
Description: Internal constructor for the `smoothKSpace` power spectrum window function class.
Code lines: 10
Contained by: file `structure_formation.power_spectrum.variance.window_function.smooth_k_space.F90`
Modules used: `numerical_constants_math`

function: `smoothkspaceconstructorparameters`
Description: Constructor for the `smoothKSpace` power spectrum window function class which takes a parameter set as input.
Code lines: 34
Contained by: file `structure_formation.power_spectrum.variance.window_function.smooth_k_space.F90`
Modules used: `input_parameters`

subroutine: `smoothkspacedestructor`
Description: Destructor for the `smoothKSpace` power spectrum window function class.
Code lines: 7
Contained by: file `structure_formation.power_spectrum.variance.window_function.smooth_k_space.F90`

function: `smoothkspacevalue`
Description: Smooth-k space power spectrum window function proposed in Leo et al. [2018]. spectrum.
Code lines: 19
Contained by: file `structure_formation.power_spectrum.variance.window_function.smooth_k_space.F90`
Modules used: `cosmology_parameters` `numerical_constants_math`

function: `smoothkspacewavenumbermaximum`
Description: Maximum wavenumber for a top hat in real space window function Fourier transformed into k-space used in computing the variance of the power spectrum.
Code lines: 11
Contained by: file `structure_formation.power_spectrum.variance.window_function.smooth_k_space.F90`

file: `structure_formation.power_spectrum.variance.window_function.top_hat.F90`
Description: Contains a module which implements a top-hat power spectrum window function class.
Code lines: 115
Modules used: `cosmology_parameters`

interface: `powerspectrumwindowfunctiontophat`
Description: Constructors for the `topHat` power spectrum window function class.
Code lines: 4
Contained by: file `structure_formation.power_spectrum.variance.window_function.top_hat.F90`

function: `tophatconstructorinternal`
Description: Internal constructor for the `topHat` power spectrum window function class.
Code lines: 8
Contained by: file `structure_formation.power_spectrum.variance.window_function.top_hat.F90`

function: `tophatconstructorparameters`
Description: Constructor for the `topHat` power spectrum window function class which takes a parameter set as input.
Code lines: 13
Contained by: file `structure_formation.power_spectrum.variance.window_function.top_hat.F90`
Modules used: `input_parameters`

subroutine: `tophatdestructor`
Description: Destructor for the `topHat` power spectrum window function class.
Code lines: 7
Contained by: file `structure_formation.power_spectrum.variance.window_function.top_hat.F90`

function: `tophatvalue`
Description: Top hat in real space window function Fourier transformed into k-space used in computing the variance of the power spectrum.
Code lines: 24
Contained by: file `structure_formation.power_spectrum.variance.window_function.top_hat.F90`
Modules used: `cosmology_parameters` `numerical_constants_math`

function: `tophatwavenumbermaximum`
Description: Maximum wavenumber for a top hat in real space window function Fourier transformed into k-space used in computing the variance of the power spectrum.
Code lines: 11
Contained by: file `structure_formation.power_spectrum.variance.window_function.top_hat.F90`

file: `structure_formation.power_spectrum.variance.window_function.top_hat_kspace_sharp_-hybrid.F90`

Description: Contains a module which implements a hybrid top-hat/sharp k-space power spectrum window function class.
Code lines: 214
Modules used: cosmology_parameters

interface: powerspectrumwindowfunctiontophatsharpkhybrid
Description: Constructors for the topHatSharpKHybrid power spectrum window function class.
Code lines: 4
Contained by: file structure_formation.power_spectrum.variance.window_function.top_hat_kspace_sharp_hybri

function: tophatsharpkhybridconstructorinternal
Description: Internal constructor for the topHatSharpKHybrid power spectrum window function class.
Code lines: 22
Contained by: file structure_formation.power_spectrum.variance.window_function.top_hat_kspace_sharp_hybri
Modules used: numerical_constants_math

function: tophatsharpkhybridconstructorparameters
Description: Constructor for the topHatSharpKHybrid power spectrum window function class which takes a parameter set as input.
Code lines: 48
Contained by: file structure_formation.power_spectrum.variance.window_function.top_hat_kspace_sharp_hybri
Modules used: input_parameters

subroutine: tophatsharpkhybriddestructor
Description: Destructor for the topHatSharpKHybrid power spectrum window function class.
Code lines: 7
Contained by: file structure_formation.power_spectrum.variance.window_function.top_hat_kspace_sharp_hybri

subroutine: tophatsharpkhybridradii
Description: Computes the radii of the top-hat and sharp k-space filters. Specifically, uses a convolution of top-hat real-space and sharp k-space window functions. The top-hat radius is $r_{\rm th}$, while the k-space cut-off wavenumber is $k_{\rm s} = a/r_{\rm s}$, where $a =$[normalization]. The two radii are chosen such that $r_{\rm th}^2 + r_{\rm s}^2 = (3M/4\pi\bar{rho})^{1/3}$ and $r_{\rm s} = \beta r_{\rm th}$ where $\beta =$[radiiRatio].
Code lines: 18
Contained by: file structure_formation.power_spectrum.variance.window_function.top_hat_kspace_sharp_hybri
Modules used: numerical_constants_math

function: tophatsharpkhybridvalue
Description: Computes a window function for calculations of the variance in the power spectrum. Specifically, uses a convolution of top-hat real-space and sharp k-space window functions. The top-hat radius is $r_{\rm th}$, while the k-space cut-off wavenumber is $k_{\rm s} = a/r_{\rm s}$, where $a =$[normalization]. The two radii are chosen such that $r_{\rm th}^2 + r_{\rm s}^2 = (3M/4\pi\bar{rho})^{1/3}$ and $r_{\rm s} = \beta r_{th}$ where $\beta =$[pradiiRatio].
Code lines: 38
Contained by: file structure_formation.power_spectrum.variance.window_function.top_hat_kspace_sharp_hybri

function: tophatsharpkhybridwavenumbermaximum

Description: Computes the maximum wavenumber at which the window function for calculations of the variance in the power spectrum is non-zero. Specifically, uses a convolution of top-hat real-space and sharp k-space window functions. The top-hat radius is $r_{\rm th}$, while the k-space cut-off wavenumber is $k_{\rm s} = a/r_{\rm s}$, where $a =$`[normalization]`. The two radii are chosen such that $r_{\rm th}^2 + r_{\rm s}^2 = (3M/4\pi r\bar{h}o)^{1/3}$ and $r_{\rm s} = \beta r_{\rm th}$ where $\beta =$`[radiiRatio]`.

Code lines: 14

Contained by: file structure_formation.power_spectrum.variance.window_function.top_hat_kspace_sharp_hybri

file: structure_formation.spherical_collapse.matter_dark_energy.F90

Description: Contains a module which implements calculations of spherical top hat collapse in cosmologies containing matter and dark energy.

Code lines: 347

module: spherical_collapse_matter_dark_energy

Description: Implements calculations of spherical top hat collapse in cosmologies containing matter and dark energy.

Code lines: 325

Contained by: file structure_formation.spherical_collapse.matter_dark_energy.F90

Modules used: cosmology_functions, fgsl, iso_varying_string

Used by: subroutine node_component_basic_-extended_bindings, subroutine sphericalcollapsematterderetabulate, function sphericalcollapsematterdeconstructorparameters, subroutine sphericalcollapsematterderetabulate, function sphericalcollapsematterdeturnaroundovervirialradii

function: expansionrateperturbation

Description: Return the expansion rate of a spherical top-hat perturbation in a dark energy universe given an initial perturbation amplitude epsilonPerturbation.

Code lines: 8

Contained by: module spherical_collapse_matter_dark_energy

subroutine: make_table

Description: Tabulate $\delta_{\rm crit}$ or $\Delta_{\rm vir}$ vs. time.

Code lines: 159

Contained by: module spherical_collapse_matter_dark_energy

Modules used: galacticus_display, galacticus_error, input_parameters, linear_growth, root_finder, tables

subroutine: perturbation_dynamics_solver

Description: Integrate the dynamics of a spherical top-hat perturbation in a dark energy universe given an initial perturbation amplitude epsilonPerturbation.

Code lines: 44

Contained by: module spherical_collapse_matter_dark_energy

Modules used: fodeiv2, odeiv2_solver

function: perturbationodes

Code lines: 17

Contained by: module spherical_collapse_matter_dark_energy

function: radiusperturbation

Description: Return the radius of a spherical top-hat perturbation in a dark energy universe given an initial perturbation amplitude epsilonPerturbation.

Code lines: 8

Contained by: module spherical_collapse_matter_dark_energy

subroutine: spherical_collapse_dark_energy_critical_overdensity_tabulate
Description: Tabulate the critical overdensity for collapse for the spherical collapse model.
Code lines: 12
Contained by: module spherical_collapse_matter_dark_energy
Modules used: linear_growth tables

subroutine: spherical_collapse_dark_energy_turnaround_radius_tabulate
Description: Tabulate the ratio of turnaround to virial radiii for the spherical collapse model.
Code lines: 11
Contained by: module spherical_collapse_matter_dark_energy
Modules used: tables

subroutine: spherical_collapse_dark_energy_virial_density_contrast_tabulate
Description: Tabulate the virial density contrast for the spherical collapse model.
Code lines: 11
Contained by: module spherical_collapse_matter_dark_energy
Modules used: tables

file: structure_formation.spherical_collapse.matter_lambda.F90
Description: Contains a module which implements calculations of spherical top hat collapse in cosmologies containing matter and a cosmological constant.
Code lines: 560

module: spherical_collapse_matter_lambda
Code lines: 537
Contained by: file structure_formation.spherical_collapse.matter_lambda.F90
Modules used: fgsl iso_c_binding
Used by: subroutine function normaloverdensitynonlinear sphericalcollapsematterlambdaretabulate
subroutine program tests_spherical_collapse_-sphericalcollapsematterlambdaretabulate nonlinear

function: amaximumroot
Description: Root function for maximum expansion radius.
Code lines: 6
Contained by: module spherical_collapse_matter_lambda

function: collapseroot
Code lines: 7
Contained by: module spherical_collapse_matter_lambda

subroutine: make_table
Description: Tabulate $\delta_{\rm crit}$ or $\Delta_{\rm vir}$ vs. time.
Code lines: 115
Contained by: module spherical_collapse_matter_lambda
Modules used: cosmology_functions galacticus_error kind_numbers linear_growth

root_finder tables

function: `perturbation_integrand`
Code lines: 13
Contained by: module spherical_collapse_matter_lambda

function: `perturbation_maximum_radius`
Description: Find the maximum radius of a perturbation with initial curvature `epsilonPerturbation`.
Code lines: 37
Contained by: module spherical_collapse_matter_lambda
Modules used: root_finder

function: `radius_root`
Code lines: 19
Contained by: module spherical_collapse_matter_lambda
Modules used: numerical_integration

subroutine: `restore_table`
Description: Attempt to restore a table from file.
Code lines: 44
Contained by: module spherical_collapse_matter_lambda
Modules used: file_utilities, galacticus_error, io_hdf5, iso_varying_string, tables

subroutine: `spherical_collape_matter_lambda_delta_virial_tabulate`
Description: Tabulate the virial density contrast for the spherical collapse model.
Code lines: 22
Contained by: module spherical_collapse_matter_lambda
Modules used: cosmology_functions, galacticus_error, galacticus_paths, iso_varying_string, tables

subroutine: `spherical_collapse_matter_lambda_critical_overdensity_tabulate`
Description: Tabulate the critical overdensity for collapse for the spherical collapse model.
Code lines: 24
Contained by: module spherical_collapse_matter_lambda
Modules used: cosmology_functions, galacticus_error, galacticus_paths, iso_varying_string, linear_growth, tables

subroutine: `spherical_collapse_matter_lambda_nonlinear_mapping`
Description: Tabulate the critical overdensity for collapse for the spherical collapse model.
Code lines: 153
Contained by: module spherical_collapse_matter_lambda
Modules used: array_utilities, arrays_search, cosmology_functions, linear_growth, numerical_integration, numerical_ranges

root_finder table_labels
tables

subroutine: store_table
Description: Attempt to restore a table from file.
Code lines: 25
Contained by: module spherical_collapse_matter_lambda
Modules used: file_utilities io_hdf5
iso_varying_string tables

function: tcollapse
Code lines: 29
Contained by: module spherical_collapse_matter_lambda
Modules used: numerical_integration

file: structure_formation.transfer_function.BBKS.F90
Description: Contains a module which implements the transfer function fitting function of Bardeen et al. [1986].
Code lines: 163
Modules used: cosmology_parameters dark_matter_particles

function: bbksconstructorinternal
Description: Internal constructor for the "BBKS" transfer function class.
Code lines: 22
Contained by: file structure_formation.transfer_function.BBKS.F90
Modules used: galacticus_error

function: bbksconstructorparameters
Description: Constructor for the "BBKS" transfer function class which takes a parameter set as input.
Code lines: 20
Contained by: file structure_formation.transfer_function.BBKS.F90
Modules used: cosmology_functions input_parameters

subroutine: bbksdestructor
Description: Destructor for the "BBKS" transfer function class.
Code lines: 8
Contained by: file structure_formation.transfer_function.BBKS.F90

function: bbksepochtime
Description: Return the cosmic time at the epoch at which this transfer function is defined.
Code lines: 7
Contained by: file structure_formation.transfer_function.BBKS.F90

function: bbkshalfmodemass
Description: Compute the mass corresponding to the wavenumber at which the transfer function is suppressed by a factor of two relative to a CDM transfer function. Not supported in this implementation.
Code lines: 16
Contained by: file structure_formation.transfer_function.BBKS.F90
Modules used: galacticus_error

function: `bbkslogarithmicderivative`
Description: Return the logarithmic derivative of the transfer function at the given wavenumber.
Code lines: 12
Contained by: file `structure_formation.transfer_function.BBKS.F90`

function: `bbksvalue`
Description: Return the transfer function at the given wavenumber.
Code lines: 12
Contained by: file `structure_formation.transfer_function.BBKS.F90`

interface: `transferfunctionbbks`
Description: Constructors for the "BBKS" transfer function class.
Code lines: 4
Contained by: file `structure_formation.transfer_function.BBKS.F90`

file: `structure_formation.transfer_function.BBKS.WDM.F90`
Description: Contains a module which implements a transfer function class based on the WDM modifier of Bardeen et al. [1986].
Code lines: 162
Modules used: `cosmology_parameters` `dark_matter_particles`

function: `bbkswdmconstructorinternal`
Description: Internal constructor for the "`bbksWDM`" transfer function class.
Code lines: 25
Contained by: file `structure_formation.transfer_function.BBKS.WDM.F90`
Modules used: `galacticus_error`

function: `bbkswdmconstructorparameters`
Description: Constructor for the "`bbksWDM`" transfer function class which takes a parameter set as input.
Code lines: 22
Contained by: file `structure_formation.transfer_function.BBKS.WDM.F90`
Modules used: `input_parameters`

subroutine: `bbkswdmdestructor`
Description: Destructor for the `bbksWDM` transfer function class.
Code lines: 9
Contained by: file `structure_formation.transfer_function.BBKS.WDM.F90`

function: `bbkswdmepochtime`
Description: Return the cosmic time at the epoch at which this transfer function is defined.
Code lines: 7
Contained by: file `structure_formation.transfer_function.BBKS.WDM.F90`

function: `bbkswdmhalfmodemass`
Description: Compute the mass corresponding to the wavenumber at which the transfer function is suppressed by a factor of two relative to a CDM transfer function.
Code lines: 16
Contained by: file `structure_formation.transfer_function.BBKS.WDM.F90`
Modules used: `galacticus_error` `numerical_constants_math`

function: `bbkswdmlogarithmicderivative`
Description: Return the logarithmic derivative of the transfer function at the given wavenumber.
Code lines: 10
Contained by: file `structure_formation.transfer_function.BBKS.WDM.F90`

function: `bbkswdmvalue`
Description: Return the transfer function at the given wavenumber.
Code lines: 10
Contained by: file `structure_formation.transfer_function.BBKS.WDM.F90`

interface: `transferfunctionbbkswdm`
Description: Constructors for the "`bbksWDM`" transfer function class.
Code lines: 4
Contained by: file `structure_formation.transfer_function.BBKS.WDM.F90`

file: `structure_formation.transfer_function.Bode2001.F90`
Description: Contains a module which implements a transfer function class based on the thermal WDM modifier of Bode et al. [2001].
Code lines: 199
Modules used: `cosmology_parameters` `dark_matter_particles`

function: `bode2001constructorinternal`
Description: Internal constructor for the "`bode2001`" transfer function class.
Code lines: 22
Contained by: file `structure_formation.transfer_function.Bode2001.F90`
Modules used: `galacticus_error`

function: `bode2001constructorparameters`
Description: Constructor for the "`bode2001`" transfer function class which takes a parameter set as input.
Code lines: 64
Contained by: file `structure_formation.transfer_function.Bode2001.F90`
Modules used: `cosmology_functions` `cosmology_functions_parameters` `galacticus_error` `input_parameters`

subroutine: `bode2001destructor`
Description: Destructor for the bode2001 transfer function class.
Code lines: 9
Contained by: file `structure_formation.transfer_function.Bode2001.F90`

function: `bode2001epochtime`
Description: Return the cosmic time at the epoch at which this transfer function is defined.
Code lines: 7
Contained by: file `structure_formation.transfer_function.Bode2001.F90`

function: `bode2001halfmodemass`
Description: Compute the mass corresponding to the wavenumber at which the transfer function is suppressed by a factor of two relative to a CDM transfer function.
Code lines: 16
Contained by: file `structure_formation.transfer_function.Bode2001.F90`

Modules used: galacticus_error numerical_constants_math

function: bode2001logarithmicderivative
Description: Return the logarithmic derivative of the transfer function at the given wavenumber.
Code lines: 9
Contained by: file structure_formation.transfer_function.Bode2001.F90

function: bode2001value
Description: Return the transfer function at the given wavenumber.
Code lines: 9
Contained by: file structure_formation.transfer_function.Bode2001.F90

interface: transferfunctionbode2001
Description: Constructors for the "bode2001" transfer function class.
Code lines: 4
Contained by: file structure_formation.transfer_function.Bode2001.F90

file: structure_formation.transfer_function.CAMB.F90
Description: Contains a module which implements a transfer function class using the CAMB code.
Code lines: 221
Modules used: cosmology_parameters dark_matter_particles file_utilities tables

subroutine: cambcheckrange
Description: Check that the provided wavenumber is within the tabulated range and, if not, recompute the CAMB transfer function.
Code lines: 52
Contained by: file structure_formation.transfer_function.CAMB.F90
Modules used: interfaces_camb

function: cambconstructorinternal
Description: Internal constructor for the CAMB transfer function class.
Code lines: 30
Contained by: file structure_formation.transfer_function.CAMB.F90
Modules used: dark_matter_particles galacticus_error input_parameters numerical_constants_astronomical

function: cambconstructorparameters
Description: Constructor for the CAMB transfer function class which takes a parameter set as input.
Code lines: 34
Contained by: file structure_formation.transfer_function.CAMB.F90
Modules used: input_parameters

subroutine: cambdestructor
Description: Destructor for the CAMB transfer function class.
Code lines: 8
Contained by: file structure_formation.transfer_function.CAMB.F90

function: camblogarithmicderivative
Description: Return the logarithmic derivative of the transfer function at the given wavenumber.

Code lines: 9
Contained by: file structure_formation.transfer_function.CAMB.F90

function: cambvalue
Description: Return the transfer function at the given wavenumber.
Code lines: 9
Contained by: file structure_formation.transfer_function.CAMB.F90

interface: transferfunctioncamb
Description: Constructors for the file transfer function class.
Code lines: 4
Contained by: file structure_formation.transfer_function.CAMB.F90

file: structure_formation.transfer_function.Eisenstein_Hu.F90
Description: Contains a module which implements a transfer function class using the fitting function of Eisenstein and Hu [1999].
Code lines: 271
Modules used: cosmology_parameters dark_matter_particles

subroutine: eisensteinhu1999computefactors
Description: Compute common factors required by "eisensteinHu1999" transfer function class.
Code lines: 26
Contained by: file structure_formation.transfer_function.Eisenstein_Hu.F90

function: eisensteinhu1999constructorinternal
Description: Internal constructor for the "eisensteinHu1999" transfer function class.
Code lines: 53
Contained by: file structure_formation.transfer_function.Eisenstein_Hu.F90
Modules used: dark_matter_particles galacticus_error

function: eisensteinhu1999constructorparameters
Description: Constructor for the "eisensteinHu1999" transfer function class which takes a parameter set as input.
Code lines: 41
Contained by: file structure_formation.transfer_function.Eisenstein_Hu.F90
Modules used: cosmology_functions input_parameters

subroutine: eisensteinhu1999destructor
Description: Destructor for the eisensteinHu1999 transfer function class.
Code lines: 8
Contained by: file structure_formation.transfer_function.Eisenstein_Hu.F90

function: eisensteinhu1999epochtime
Description: Return the cosmic time at the epoch at which this transfer function is defined.
Code lines: 7
Contained by: file structure_formation.transfer_function.Eisenstein_Hu.F90

function: eisensteinhu1999halfmodemass
Description: Compute the mass corresponding to the wavenumber at which the transfer function is suppressed by a factor of two relative to a CDM transfer function. Not supported in this implementation.
Code lines: 16

Contained by: file structure_formation.transfer_function.Eisenstein_Hu.F90
Modules used: galacticus_error

function: eisensteinhu1999logarithmicderivative
Description: Return the logarithmic derivative of the transfer function at the given wavenumber.
Code lines: 25
Contained by: file structure_formation.transfer_function.Eisenstein_Hu.F90

function: eisensteinhu1999value
Description: Return the transfer function at the given wavenumber.
Code lines: 17
Contained by: file structure_formation.transfer_function.Eisenstein_Hu.F90

interface: transferfunctioneisensteinhu1999
Description: Constructors for the "eisensteinHu1999" transfer function class.
Code lines: 4
Contained by: file structure_formation.transfer_function.Eisenstein_Hu.F90

file: structure_formation.transfer_function.F90
Description: Contains a module which provides a class that implements transfer functions.
Code lines: 56

module: transfer_functions
Description: Provides an object that implements transfer functions.
Code lines: 34
Contained by: file structure_formation.transfer_function.F90
Used by: file dark_matter_profiles.structure.concentration.WDM.F90; subroutine galacticus_function_classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; file structure_formation.power_spectrum.primordial.transferred.simple.F90; file tasks.halo_mass_function.F90; file tasks.merger_tree_file_builder.F90; program tests_power_spectrum; program tests_sigma; program tests_transfer_functions

file: structure_formation.transfer_function.accelerator.F90
Description: Implements a transfer function accelerator class which tabulates a transfer function for rapid interpolation.
Code lines: 167
Modules used: tables

function: acceleratorconstructorinternal
Description: Internal constructor for the accelerator transfer function class.
Code lines: 12
Contained by: file structure_formation.transfer_function.accelerator.F90

function: acceleratorconstructorparameters
Description: Constructor for the accelerator transfer function class which takes a parameter set as input.
Code lines: 21
Contained by: file structure_formation.transfer_function.accelerator.F90

subroutine: `acceleratordestructor`
Description: Destructor for the accelerator transfer function class.
Code lines: 8
Contained by: file `structure_formation.transfer_function.accelerator.F90`

function: `acceleratorepochtime`
Description: Return the cosmic time at the epoch at which this transfer function is defined.
Code lines: 7
Contained by: file `structure_formation.transfer_function.accelerator.F90`

function: `acceleratorhalfmodemass`
Description: Compute the mass corresponding to the wavenumber at which the transfer function is suppressed by a factor of two relative to a CDM transfer function
Code lines: 9
Contained by: file `structure_formation.transfer_function.accelerator.F90`

function: `acceleratorlogarithmicderivative`
Description: Return the logarithmic derivative of the transfer function at the given wavenumber.
Code lines: 11
Contained by: file `structure_formation.transfer_function.accelerator.F90`

subroutine: `acceleratortabulate`
Description: Tabulate the transfer function for rapid interpolation.
Code lines: 21
Contained by: file `structure_formation.transfer_function.accelerator.F90`

function: `acceleratorvalue`
Description: Return the transfer function at the given wavenumber.
Code lines: 11
Contained by: file `structure_formation.transfer_function.accelerator.F90`

interface: `transferfunctionaccelerator`
Description: Constructors for the accelerator transfer function class.
Code lines: 4
Contained by: file `structure_formation.transfer_function.accelerator.F90`

file: `structure_formation.transfer_function.file.F90`
Description: Implements a file-based transfer function class.
Code lines: 343
Modules used: `cosmology_functions` `cosmology_parameters` `tables`

function: `fileconstructorinternal`
Description: Internal constructor for the file transfer function class.
Code lines: 13
Contained by: file `structure_formation.transfer_function.file.F90`

function: `fileconstructorparameters`
Description: Constructor for the file transfer function class which takes a parameter set as input.

Code lines: 33
Contained by: file structure_formation.transfer_function.file.F90
Modules used: input_parameters

subroutine: filedestructor
Description: Destructor for the file transfer function class.
Code lines: 9
Contained by: file structure_formation.transfer_function.file.F90

function: fileepochtime
Description: Return the cosmic time at the epoch at which this transfer function is defined.
Code lines: 7
Contained by: file structure_formation.transfer_function.file.F90

function: filehalfmodemass
Description: Compute the mass corresponding to the wavenumber at which the transfer function is suppressed by a factor of two relative to a CDM transfer function. Not supported in this implementation.
Code lines: 17
Contained by: file structure_formation.transfer_function.file.F90
Modules used: galacticus_error

function: filelogarithmicderivative
Description: Return the logarithmic derivative of the transfer function at the given wavenumber.
Code lines: 8
Contained by: file structure_formation.transfer_function.file.F90

subroutine: filereadfile
Description: Internal constructor for the file transfer function class.
Code lines: 74
Contained by: file structure_formation.transfer_function.file.F90
Modules used: array_utilities, fgsl, file_utilities, galacticus_display, galacticus_error, input_parameters, io_hdf5, numerical_comparison, table_labels

function: filevalue
Description: Return the transfer function at the given wavenumber.
Code lines: 8
Contained by: file structure_formation.transfer_function.file.F90

interface: transferfunctionfile
Description: Constructors for the file transfer function class.
Code lines: 4
Contained by: file structure_formation.transfer_function.file.F90

file: structure_formation.transfer_function.identity.F90
Description: Contains a module which implements an identity transfer function class.
Code lines: 138

function: `identityconstructorinternal`
Description: Internal constructor for the identity transfer function class.
Code lines: 9
Contained by: file `structure_formation.transfer_function.identity.F90`
Modules used: `input_parameters`

function: `identityconstructorparameters`
Description: Constructor for the identity transfer function class which takes a parameter set as input.
Code lines: 23
Contained by: file `structure_formation.transfer_function.identity.F90`
Modules used: `cosmology_functions` `input_parameters`

subroutine: `identitydestructor`
Description: Destructor for the identity transfer function class.
Code lines: 8
Contained by: file `structure_formation.transfer_function.identity.F90`

function: `identityepochtime`
Description: Return the cosmic time at the epoch at which this transfer function is defined.
Code lines: 7
Contained by: file `structure_formation.transfer_function.identity.F90`

function: `identityhalfmodemass`
Description: Compute the mass corresponding to the wavenumber at which the transfer function is suppressed by a factor of two relative to a CDM transfer function. Not supported in this implementation.
Code lines: 16
Contained by: file `structure_formation.transfer_function.identity.F90`
Modules used: `galacticus_error`

function: `identitylogarithmicderivative`
Description: Return the logarithmic derivative of the transfer function at the given wavenumber.
Code lines: 9
Contained by: file `structure_formation.transfer_function.identity.F90`

function: `identityvalue`
Description: Return the transfer function at the given wavenumber.
Code lines: 9
Contained by: file `structure_formation.transfer_function.identity.F90`

interface: `transferfunctionidentity`
Description: Constructors for the identity transfer function class.
Code lines: 4
Contained by: file `structure_formation.transfer_function.identity.F90`

file: `structure_formation.unevolved_subhalo_mass_function.F90`
Description: Contains a module which provides a class that implements subhalo mass functions.
Code lines: 45

module: `unevolved_subhalo_mass_functions`
Code lines: 23
Contained by: file `structure_formation.unevolved_subhalo_mass_function.F90`
Used by: subroutine `galacticus_function_-classes_destroy` subroutine `galacticus_state_retrieve` subroutine `galacticus_state_store` file `tasks.halo_mass_function.F90`

file: `structure_formation.unevolved_subhalo_mass_function.Giocoli2008.F90`
Description: Contains a module which implements a Giocoli et al. [2008] unevolved dark matter subhalo mass function class.
Code lines: 135

function: `giocoli2008constructorinternal`
Description: Internal constructor for the `giocoli2008` halo mass function class.
Code lines: 9
Contained by: file `structure_formation.unevolved_subhalo_mass_function.Giocoli2008.F90`

function: `giocoli2008constructorparameters`
Description: Constructor for the `giocoli2008` halo mass function class which takes a parameter set as input.
Code lines: 30
Contained by: file `structure_formation.unevolved_subhalo_mass_function.Giocoli2008.F90`
Modules used: `input_parameters`

function: `giocoli2008differential`
Description: Return the differential unevolved subhalo mass function at the given time and mass.
Code lines: 11
Contained by: file `structure_formation.unevolved_subhalo_mass_function.Giocoli2008.F90`

function: `giocoli2008integrated`
Description: Return the integrated unevolved subhalo mass function at the given time and mass.
Code lines: 33
Contained by: file `structure_formation.unevolved_subhalo_mass_function.Giocoli2008.F90`
Modules used: `gamma_functions`

function: `gammaincomplete`
Description: Evaluate the incomplete gamma function, possibly for a negative exponent.
Code lines: 15
Contained by: function `giocoli2008integrated`
Modules used: `galacticus_error`

interface: `unevolvedsubhalomassfunctiongiocoli2008`
Description: Constructors for the `giocoli2008` halo mass function class.
Code lines: 4
Contained by: file `structure_formation.unevolved_subhalo_mass_function.Giocoli2008.F90`

file: `structure_formation.virial_density_contrast.Bryan_Norman.F90`
Description: An implementation of Bryan and Norman [1998] dark matter halo virial density contrasts.

Code lines: 171
Modules used: cosmology_functions cosmology_parameters

function: bryannorman1998constructorinternal
Description: Internal constructor for the bryanNorman1998 dark matter halo virial density contrast class.
Code lines: 19
Contained by: file structure_formation.virial_density_contrast.Bryan_Norman.F90
Modules used: galacticus_error numerical_comparison

function: bryannorman1998constructorparameters
Description: Constructor for the bryanNorman1998 dark matter halo virial density contrast class which takes a parameter set as input.
Code lines: 16
Contained by: file structure_formation.virial_density_contrast.Bryan_Norman.F90
Modules used: input_parameters

function: bryannorman1998densitycontrast
Description: Return the virial density contrast at the given epoch, assuming the fitting function of Bryan and Norman [1998].
Code lines: 24
Contained by: file structure_formation.virial_density_contrast.Bryan_Norman.F90
Modules used: cosmology_functions galacticus_error numerical_constants_math

function: bryannorman1998densitycontrastrateofchange
Description: Return the virial density contrast at the given epoch, assuming the fitting function of Bryan and Norman [1998].
Code lines: 24
Contained by: file structure_formation.virial_density_contrast.Bryan_Norman.F90
Modules used: cosmology_functions galacticus_error numerical_constants_math

subroutine: bryannorman1998destructor
Description: Destructor for the bryanNorman1998 virial density contrast class.
Code lines: 8
Contained by: file structure_formation.virial_density_contrast.Bryan_Norman.F90

function: bryannorman1998turnaroundovervirialradii
Description: Return the ratio of turnaround and virial radii at the given epoch, based spherical collapse in a matter plus cosmological constant universe.
Code lines: 13
Contained by: file structure_formation.virial_density_contrast.Bryan_Norman.F90

interface: virialdensitycontrastbryannorman1998
Description: Constructors for the bryanNorman1998 dark matter halo virial density contrast class.
Code lines: 4
Contained by: file structure_formation.virial_density_contrast.Bryan_Norman.F90

file: structure_formation.virial_density_contrast.F90

Description: Contains a module which provides a class implementing the virial density contrast for halos.
Code lines: 72

module: `virial_density_contrast`
Description: Provides a class implementing the virial density contrast for halos.
Code lines: 50
Contained by: file structure_formation.virial_density_contrast.F90
Modules used: iso_varying_string
Used by:

file dark_matter_halos.mass_loss_rates.vanDenBosch.F90	file dark_matter_halos.scales.virial_density_contrast.F90
function bullock2001concentration	module dark_matter_profiles_concentration
file dark_matter_profiles.structure.concentration.NFW.F90	function nfw1996concentration
function dark_matter_profile_mass_definition	file dark_matter_profiles.structure.scale.concentration.F90
file galactic.filters.halo_mass.F90	subroutine galacticus_function_classes_destroy
function hivshalomassrelationpadmanabhan2017constructorinternal	function concentrationvshalomassdmcococonstructorinternal
function concentrationvshalomassdmludlow2016constructorinternal	function spindistributionbett2007constructorinternal
function stellarvshalomassrelationleauthaud2012constructorinternal	subroutine galacticus_state_retrieve
subroutine galacticus_state_store	file merger_trees.operators.prune_baryons.F90
file nodes.property_extractor.concentration.F90	file nodes.property_extractor.mass_halo.F90
module node_component_basic_standard_extended	module virial_orbits
file structure_formation.halo_bias.Tinker2010.F90	file structure_formation.halo_mass_function.Despali_2015.F90
file structure_formation.halo_mass_function.Tinker2008.F90	function virial_density_contrast_percolation_solver
file tasks.halo_mass_function.F90	program tests_spherical_collapse_dark_energy_eds
program tests_spherical_collapse_dark_energy_omega_zero_point_six	program tests_spherical_collapse_dark_energy_omega_zero_point_eight
program tests_spherical_collapse_dark_energy_lambda	program tests_spherical_collapse_flat
program tests_spherical_collapse_open	

file: `structure_formation.virial_density_contrast.Kitayama_Suto1996.F90`
Description: An implementation of Kitayama and Suto [1996] dark matter halo virial density contrasts.
Code lines: 111
Modules used: cosmology_functions

function: `kitayamasuto1996constructorinternal`

Description: Internal constructor for the `kitayamaSuto1996` dark matter halo virial density contrast class.
Code lines: 10
Contained by: file `structure_formation.virial_density_contrast.Kitayama_Suto1996.F90`
Modules used: `galacticus_error` `numerical_comparison`

function: `kitayamasuto1996constructorparameters`
Description: Constructor for the `kitayamaSuto1996` dark matter halo virial density contrast class which takes a parameter set as input.
Code lines: 13
Contained by: file `structure_formation.virial_density_contrast.Kitayama_Suto1996.F90`
Modules used: `input_parameters`

function: `kitayamasuto1996densitycontrast`
Description: Return the virial density contrast at the given epoch, assuming the fitting function of Kitayama and Suto [1996].
Code lines: 14
Contained by: file `structure_formation.virial_density_contrast.Kitayama_Suto1996.F90`
Modules used: `numerical_constants_math`

function: `kitayamasuto1996densitycontrastrateofchange`
Description: Return the virial density contrast at the given epoch, assuming the fitting function of Kitayama and Suto [1996].
Code lines: 14
Contained by: file `structure_formation.virial_density_contrast.Kitayama_Suto1996.F90`
Modules used: `numerical_constants_math`

subroutine: `kitayamasuto1996destructor`
Description: Destructor for the `bryanNorman1998` virial density contrast class.
Code lines: 7
Contained by: file `structure_formation.virial_density_contrast.Kitayama_Suto1996.F90`

interface: `virialdensitycontrastkitayamasuto1996`
Description: Constructors for the `kitayamaSuto1996` dark matter halo virial density contrast class.
Code lines: 4
Contained by: file `structure_formation.virial_density_contrast.Kitayama_Suto1996.F90`

file: `structure_formation.virial_density_contrast.fixed.F90`
Description: An implementation of fixed dark matter halo virial density contrasts.
Code lines: 166
Modules used: `cosmology_functions` `cosmology_parameters`

function: `fixedconstructorinternal`
Description: Constructor for the `fixed` dark matter halo virial density contrast class.
Code lines: 13
Contained by: file `structure_formation.virial_density_contrast.fixed.F90`
Modules used: `galacticus_error`

function: `fixedconstructorparameters`

Description: Constructor for the `fixed` dark matter halo virial density contrast class that takes a parameter set as input.
Code lines: 35
Contained by: file `structure_formation.virial_density_contrast.fixed.F90`
Modules used: `input_parameters` `iso_varying_string`

function: `fixeddensitycontrast`
Description: Return the virial density contrast at the given epoch, assuming a fixed contrast.
Code lines: 23
Contained by: file `structure_formation.virial_density_contrast.fixed.F90`

function: `fixeddensitycontrastrateofchange`
Description: Return the virial density contrast at the given epoch, assuming a fixed contrast.
Code lines: 19
Contained by: file `structure_formation.virial_density_contrast.fixed.F90`

subroutine: `fixeddestructor`
Description: Destructor for the `fixed` virial density contrast class.
Code lines: 8
Contained by: file `structure_formation.virial_density_contrast.fixed.F90`

interface: `virialdensitycontrastfixed`
Description: Constructors for the `fixed` dark matter halo virial density contrast class.
Code lines: 4
Contained by: file `structure_formation.virial_density_contrast.fixed.F90`

file: `structure_formation.virial_density_contrast.friends-of-friends.F90`
Description: An implementation of dark matter halo virial density contrasts based on a friends-of-friends linking length.
Code lines: 110

function: `friendsoffriendsconstructorinternal`
Description: Generic constructor for the `friendsOfFriends` dark matter halo virial density contrast class.
Code lines: 9
Contained by: file `structure_formation.virial_density_contrast.friends-of-friends.F90`
Modules used: `input_parameters`

function: `friendsoffriendsconstructorparameters`
Description: Default constructor for the `friendsOfFriends` dark matter halo virial density contrast class.
Code lines: 27
Contained by: file `structure_formation.virial_density_contrast.friends-of-friends.F90`
Modules used: `input_parameters`

function: `friendsoffriendsdensitycontrast`
Description: Return the virial density contrast at the given epoch, based on the friends-of-friends algorithm linking length.
Code lines: 14
Contained by: file `structure_formation.virial_density_contrast.friends-of-friends.F90`
Modules used: `numerical_constants_math`

function: `friendsoffriendsdensitycontrastrateofchange`
Description: Return the virial density contrast at the given epoch, based on the friends-of-friends algorithm linking length.
Code lines: 12
Contained by: file `structure_formation.virial_density_contrast.friends-of-friends.F90`
Modules used: `numerical_constants_math`

interface: `virialdensitycontrastfriendsoffriends`
Description: Constructors for the `friendsOfFriends` dark matter halo virial density contrast class.
Code lines: 4
Contained by: file `structure_formation.virial_density_contrast.friends-of-friends.F90`

file: `structure_formation.virial_density_contrast.percolation.F90`
Description: An implementation of dark matter halo virial density contrasts based on the percolation analysis of More et al. [2011].
Code lines: 496
Modules used: `cosmology_functions` `kind_numbers` `tables`

function: `percolationconstructorinternal`
Description: Internal constructor for the `percolation` dark matter halo virial density contrast class.
Code lines: 57
Contained by: file `structure_formation.virial_density_contrast.percolation.F90`
Modules used: `galacticus_error` `input_parameters`

function: `percolationconstructorparameters`
Description: Constructor for the `percolation` dark matter halo virial density contrast class that takes a parameter set as input.
Code lines: 38
Contained by: file `structure_formation.virial_density_contrast.percolation.F90`
Modules used: `cosmology_functions` `functions_global` `input_parameters`

subroutine: `percolationcopytable`
Description: Copy the table from a recursive child's parent.
Code lines: 23
Contained by: file `structure_formation.virial_density_contrast.percolation.F90`
Modules used: `galacticus_error`

subroutine: `percolationdeepcopy`
Description: Perform a deep copy of the object.
Code lines: 53
Contained by: file `structure_formation.virial_density_contrast.percolation.F90`
Modules used: `functions_global` `galacticus_error`

subroutine: `percolationdeepcopyassign`
Description: Perform pointer assignment during a deep copy of the object.
Code lines: 11

Contained by: file structure_formation.virial_density_contrast.percolation.F90

function: percolationdensitycontrast
Description: Return the virial density contrast at the given epoch, based on the percolation algorithm of More et al. [2011].
Code lines: 56
Contained by: file structure_formation.virial_density_contrast.percolation.F90
Modules used: galacticus_error solvercurrent table

function: percolationdensitycontrastrateofchange
Description: Return the rate of change of the virial density contrast at the given epoch, based on the percolation algorithm of More et al. [2011].
Code lines: 34
Contained by: file structure_formation.virial_density_contrast.percolation.F90
Modules used: galacticus_error

subroutine: percolationdestructor
Description: Destructor for the percolation virial density contrast class.
Code lines: 11
Contained by: file structure_formation.virial_density_contrast.percolation.F90
Modules used: galacticus_error

subroutine: percolationfindparent
Description: Find the deep-copied parent of a recursive child.
Code lines: 15
Contained by: file structure_formation.virial_density_contrast.percolation.F90
Modules used: galacticus_error

function: percolationismassdepdendent
Description: Specify that the percolation virial density contrast class is mass-dependent.
Code lines: 8
Contained by: file structure_formation.virial_density_contrast.percolation.F90

subroutine: percolationtabulate
Description: Tabulate virial density contrast as a function of mass and time for the percolation density contrast class.
Code lines: 72
Contained by: file structure_formation.virial_density_contrast.percolation.F90
Modules used: functions_global galacticus_display galacticus_error

interface: virialdensitycontrastpercolation
Description: Constructors for the percolation dark matter halo virial density contrast class.
Code lines: 4
Contained by: file structure_formation.virial_density_contrast.percolation.F90

file: structure_formation.virial_density_contrast.percolation.utilities.F90
Description: Contains a module of utilities needed by the percolation virial density contrast class.

Code lines: 291

module: `virial_density_contrast_percolation_utilities`
Description: Provides utilities needed by the `percolation` virial density contrast class.
Code lines: 269
Contained by: file `structure_formation.virial_density_contrast.percolation.utilities.F90`
Modules used: `cosmology_functions`, `cosmology_parameters`, `dark_matter_halo_scales`, `dark_matter_profile_scales`, `dark_matter_profiles_concentration`, `dark_matter_profiles_dmo`, `dark_matter_profiles_shape`, `galacticus_nodes`
Used by: subroutine `functions_global_set`

function: `haloradiusrootfunction`
Description: Root function used to find the radius of a halo giving the correct bounding density.
Code lines: 22
Contained by: module `virial_density_contrast_percolation_utilities`
Modules used: `galacticus_calculations_resets`, `numerical_constants_math`

type: `percolationobjects`
Description: Type used to store pointers to objects
Code lines: 10
Contained by: module `virial_density_contrast_percolation_utilities`

subroutine: `percolationobjectsdeepcopy`
Description: Perform a deep copy of percolation virial density contrast objects.
Code lines: 35
Contained by: module `virial_density_contrast_percolation_utilities`
Modules used: `galacticus_error`

subroutine: `percolationobjectsdestructor`
Description: Destruct an instance of the container type for percolation virial density contrast objects.
Code lines: 12
Contained by: module `virial_density_contrast_percolation_utilities`

type: `solverstate`
Code lines: 8
Contained by: module `virial_density_contrast_percolation_utilities`

function: `virial_density_contrast_percolation_objects_constructor`
Description: Construct an instance of the container type for percolation virial density contrast objects from a parameter structure.
Code lines: 18
Contained by: module `virial_density_contrast_percolation_utilities`
Modules used: `input_parameters`

function: `virial_density_contrast_percolation_solver`
Description: Return the virial density contrast at the given epoch, based on the percolation algorithm of More et al. [2011].
Code lines: 110
Contained by: module `virial_density_contrast_percolation_utilities`

Modules used: galacticus_calculations_resets, galacticus_error, galacticus_nodes, numerical_constants_math, root_finder, virial_density_contrast

file: structure_formation.virial_density_contrast.spherical_collapse_matter_dark_energy.F90
Description: An implementation of dark matter halo virial density contrasts based on spherical collapse in a matter plus dark energy universe.
Code lines: 170
Modules used: tables

function: sphericalcollapsematterdeconstructorinternal
Description: Internal constructor for the sphericalCollapseMatterDE dark matter halo virial density contrast class.
Code lines: 11
Contained by: file structure_formation.virial_density_contrast.spherical_collapse_matter_dark_energy.F90

function: sphericalcollapsematterdeconstructorparameters
Description: Constructor for the sphericalCollapseMatterDE dark matter halo virial density contrast class that takes a parameter set as input.
Code lines: 25
Contained by: file structure_formation.virial_density_contrast.spherical_collapse_matter_dark_energy.F90
Modules used: input_parameters, spherical_collapse_matter_dark_energy

subroutine: sphericalcollapsematterdedestructor
Description: Destructor for the sphericalCollapseMatterDE dark matter halo virial density contrast class.
Code lines: 11
Contained by: file structure_formation.virial_density_contrast.spherical_collapse_matter_dark_energy.F90

subroutine: sphericalcollapsematterderetabulate
Description: Recompute the look-up tables for virial density contrast.
Code lines: 21
Contained by: file structure_formation.virial_density_contrast.spherical_collapse_matter_dark_energy.F90
Modules used: spherical_collapse_matter_dark_energy

function: sphericalcollapsematterdeturnaroundovervirialradii
Description: Return the ratio of turnaround and virial radii at the given epoch, based spherical collapse in a matter plus cosmological constant universe.
Code lines: 46
Contained by: file structure_formation.virial_density_contrast.spherical_collapse_matter_dark_energy.F90
Modules used: galacticus_error, spherical_collapse_matter_dark_energy

interface: virialdensitycontrastsphericalcollapsematterde
Description: Constructors for the sphericalCollapseMatterDE dark matter halo virial density contrast class.
Code lines: 4
Contained by: file structure_formation.virial_density_contrast.spherical_collapse_matter_dark_energy.F90

file: structure_formation.virial_density_contrast.spherical_collapse_matter_lambda.F90
Description: An implementation of dark matter halo virial density contrasts based on spherical collapse in a matter plus cosmological constant universe.

Code lines: 222
Modules used: cosmology_functions tables

function: sphericalcollapsematterlambdaconstructorinternal
Description: Internal constructor for the sphericalCollapseMatterLambda dark matter halo virial density contrast class.
Code lines: 10
Contained by: file structure_formation.virial_density_contrast.spherical_collapse_matter_lambda.F90

function: sphericalcollapsematterlambdaconstructorparameters
Description: Constructor for the sphericalCollapseMatterLambda dark matter halo virial density contrast class that takes a parameter set as input.
Code lines: 22
Contained by: file structure_formation.virial_density_contrast.spherical_collapse_matter_lambda.F90
Modules used: input_parameters

function: sphericalcollapsematterlambdadensitycontrast
Description: Return the virial density contrast at the given epoch, based spherical collapse in a matter plus cosmological constant universe.
Code lines: 36
Contained by: file structure_formation.virial_density_contrast.spherical_collapse_matter_lambda.F90
Modules used: galacticus_error

function: sphericalcollapsematterlambdadensitycontrastrateofchange
Description: Return the virial density contrast at the given epoch, based spherical collapse in a matter plus cosmological constant universe.
Code lines: 36
Contained by: file structure_formation.virial_density_contrast.spherical_collapse_matter_lambda.F90
Modules used: galacticus_error

subroutine: sphericalcollapsematterlambdadestructor
Description: Destructor for the sphericalCollapseMatterLambda dark matter halo virial density contrast class.
Code lines: 11
Contained by: file structure_formation.virial_density_contrast.spherical_collapse_matter_lambda.F90

subroutine: sphericalcollapsematterlambdaretabulate
Description: Recompute the look-up tables for virial density contrast.
Code lines: 21
Contained by: file structure_formation.virial_density_contrast.spherical_collapse_matter_lambda.F90
Modules used: spherical_collapse_matter_lambda

function: sphericalcollapsematterlambdaturnaroundovervirialradii
Description: Return the ratio of turnaround and virial radii at the given epoch, based spherical collapse in a matter plus cosmological constant universe.
Code lines: 13
Contained by: file structure_formation.virial_density_contrast.spherical_collapse_matter_lambda.F90

interface: virialdensitycontrastsphericalcollapsematterlambda
Description: Constructors for the sphericalCollapseMatterLambda dark matter halo virial density contrast class.

Code lines: 4
Contained by: file structure_formation.virial_density_contrast.spherical_collapse_matter_lambda.F90

file: system.command.F90
Description: Contains a module which executes system commands.
Code lines: 63

module: system_command
Description: Executes system commands.
Code lines: 41
Contained by: file system.command.F90
Used by:

subroutine hopkins2007buildfile	subroutine atomiccieclOudytabulate
subroutine atomiccieclOudytabulate	subroutine galacticus_output_close_file
function geometrymangleangularpower	subroutine geometrymanglebuild
function geometrymanglesolidangle	subroutine bernardi2013sdssmanglefiles
subroutine caputi2011ukidssudsrandomsinitialize	subroutine liwhite2009sdssrandomsinitialize
subroutine martin2010alfalfarandomsinitialize	function mangleangularpower
subroutine interface_camb_initialize	subroutine interface_camb_transfer_function
subroutine interface_cloudy_initialize	subroutine interface_cloudy_cie_tabulate
subroutine interface_fsps_initialize	subroutine interface_fsps_ssps_tabulate
subroutine interface_recfast_initialize	function recfastconstructorinternal
subroutine exportoperate	function outputrootmassesconstructorinternal
subroutine differentialevolutionsimulate	subroutine particleswarmsimulate
function standardinterpolate	function farahiconstructorinternal
subroutine takahashi2011lensingdistributionconstruct	function cosmicemuvalue
program test_diemerkravtsov2014_concentration	program tests_io_xml
subroutine file_remove	

subroutine: system_command_char
Description: Executes the system command command, optionally returning the resulting status in iStatus.
Code lines: 15
Contained by: module system_command
Modules used: galacticus_error

interface: system_command_do
Code lines: 3
Contained by: module system_command

subroutine: system_command_varstr
Description: Executes the system command command, optionally returning the resulting status in iStatus.

Code lines: 9
Contained by: module system_command
Modules used: iso_varying_string

file: system.limits.F90
Description: Contains a module which sets system resoruce limits.
Code lines: 71

module: system_limits
Description: Set resource limits.
Code lines: 46
Contained by: file system.limits.F90
Modules used: iso_c_binding
Used by: program galacticus

subroutine: system_limits_set
Description: Set system resource limits.
Code lines: 23
Contained by: module system_limits
Modules used: galacticus_error input_parameters

file: system.load.F90
Description: Contains a module which reports system load averages.
Code lines: 69

module: system_load
Description: Reports system load averages.
Code lines: 47
Contained by: file system.load.F90
Used by: function subroutine evolveforestsperform evolveforestsconstructorparameters

subroutine: system_load_get
Description: Reports on system load via /proc/loadavg.
Code lines: 17
Contained by: module system_load

function: system_processor_count
Description: Return a count of the number of available processors.
Code lines: 18
Contained by: module system_load

file: tasks.F90
Description: Contains a module which provides a class that implements general tasks to be performed by Galacticus.
Code lines: 50

module: tasks
Description: Provides a class that implements general tasks to be performed by Galacticus.

Code lines: 28
Contained by: file tasks.F90
Used by: program galacticus; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; subroutine tasks_evolve_forest_construct; subroutine tasks_evolve_forest_destruct; subroutine tasks_evolve_forest_perform

file: tasks.Local_Group_database.F90
Code lines: 77

function: localgroupdatabaseparameters
Description: Constructor for the localGroupDatabase task class which takes a parameter set as input.
Code lines: 10
Contained by: file tasks.Local_Group_database.F90
Modules used: input_parameters

subroutine: localgroupdatabaseperform
Description: Update the database.
Code lines: 17
Contained by: file tasks.Local_Group_database.F90
Modules used: galacticus_display; galacticus_error; interface_local_group_db

function: localgroupdatabaserequiresoutputfile
Description: Specifies that this task does not requires the main output file.
Code lines: 8
Contained by: file tasks.Local_Group_database.F90

interface: tasklocalgroupdatabase
Description: Constructors for the localGroupDatabase task.
Code lines: 3
Contained by: file tasks.Local_Group_database.F90

file: tasks.accretion_disks.spectra.Hopkins2007.build_file.F90
Code lines: 76

function: agnspectrahopkins2008buildfileparameters
Description: Constructor for the agnSpectraHopkins2008BuildFile task class which takes a parameter set as input.
Code lines: 10
Contained by: file tasks.accretion_disks.spectra.Hopkins2007.build_file.F90
Modules used: input_parameters

subroutine: agnspectrahopkins2008buildfileperform
Description: Builds the tabulation.
Code lines: 16
Contained by: file tasks.accretion_disks.spectra.Hopkins2007.build_file.F90
Modules used: accretion_disk_spectra; galacticus_display; galacticus_error

function: agnspectrahopkins2008buildfilerequiresoutputfile
Description: Specifies that this task does not requires the main output file.
Code lines: 8
Contained by: file tasks.accretion_disks.spectra.Hopkins2007.build_file.F90

interface: taskagnspectrahopkins2008buildfile
Description: Constructors for the agnSpectraHopkins2008BuildFile task.
Code lines: 3
Contained by: file tasks.accretion_disks.spectra.Hopkins2007.build_file.F90

file: tasks.build_tool.CAMB.F90
Code lines: 77

function: buildtoolcambparameters
Description: Constructor for the buildToolCAMB task class which takes a parameter set as input.
Code lines: 10
Contained by: file tasks.build_tool.CAMB.F90
Modules used: input_parameters

subroutine: buildtoolcambperform
Description: Builds the tabulation.
Code lines: 17
Contained by: file tasks.build_tool.CAMB.F90
Modules used: galacticus_display galacticus_error interfaces_camb

function: buildtoolcambrequiresoutputfile
Description: Specifies that this task does not requires the main output file.
Code lines: 8
Contained by: file tasks.build_tool.CAMB.F90

interface: taskbuildtoolcamb
Description: Constructors for the buildToolCAMB task.
Code lines: 3
Contained by: file tasks.build_tool.CAMB.F90

file: tasks.build_tool.Cloudy.F90
Code lines: 77

function: buildtoolcloudyparameters
Description: Constructor for the buildToolCloudy task class which takes a parameter set as input.
Code lines: 10
Contained by: file tasks.build_tool.Cloudy.F90
Modules used: input_parameters

subroutine: buildtoolcloudyperform
Description: Builds the tabulation.
Code lines: 17

Contained by: file tasks.build_tool.Cloudy.F90
Modules used: galacticus_display galacticus_error interfaces_cloudy

function: buildtoolcloudyrequiresoutputfile
Description: Specifies that this task does not requires the main output file.
Code lines: 8
Contained by: file tasks.build_tool.Cloudy.F90

interface: taskbuildtoolcloudy
Description: Constructors for the buildToolCloudy task.
Code lines: 3
Contained by: file tasks.build_tool.Cloudy.F90

file: tasks.build_tool.FSPS.F90
Code lines: 77

function: buildtoolfspsparameters
Description: Constructor for the buildToolFSPS task class which takes a parameter set as input.
Code lines: 10
Contained by: file tasks.build_tool.FSPS.F90
Modules used: input_parameters

subroutine: buildtoolfspsperform
Description: Builds the tabulation.
Code lines: 17
Contained by: file tasks.build_tool.FSPS.F90
Modules used: galacticus_display galacticus_error interfaces_fsps

function: buildtoolfspsrequiresoutputfile
Description: Specifies that this task does not requires the main output file.
Code lines: 8
Contained by: file tasks.build_tool.FSPS.F90

interface: taskbuildtoolfsps
Description: Constructors for the buildToolFSPS task.
Code lines: 3
Contained by: file tasks.build_tool.FSPS.F90

file: tasks.build_tool.RecFast.F90
Code lines: 77

function: buildtoolrecfastparameters
Description: Constructor for the buildToolRecFast task class which takes a parameter set as input.
Code lines: 10
Contained by: file tasks.build_tool.RecFast.F90
Modules used: input_parameters

subroutine: buildtoolrecfastperform
Description: Builds the tabulation.
Code lines: 17
Contained by: file tasks.build_tool.RecFast.F90
Modules used: galacticus_display galacticus_error interfaces_recfast

function: buildtoolrecfastrequiresoutputfile
Description: Specifies that this task does not requires the main output file.
Code lines: 8
Contained by: file tasks.build_tool.RecFast.F90

interface: taskbuildtoolrecfast
Description: Constructors for the buildToolRecFast task.
Code lines: 3
Contained by: file tasks.build_tool.RecFast.F90

file: tasks.catalog_projected_correlation_function.F90
Code lines: 406
Modules used: cosmology_functions cosmology_parameters geometry_surveys

function: catalogprojectedcorrelationfunctionconstructorinternal
Description: Constructor for the catalogProjectedCorrelationFunction task class which takes a parameter set as input.
Code lines: 16
Contained by: file tasks.catalog_projected_correlation_function.F90

function: catalogprojectedcorrelationfunctionconstructorparameters
Description: Constructor for the catalogProjectedCorrelationFunction task class which takes a parameter set as input.
Code lines: 160
Contained by: file tasks.catalog_projected_correlation_function.F90
Modules used: galacticus_nodes input_parameters node_components numerical_constants_math pseudo_random

subroutine: catalogprojectedcorrelationfunctiondestructor
Description: Destructor for the catalogProjectedCorrelationFunction task class.
Code lines: 12
Contained by: file tasks.catalog_projected_correlation_function.F90
Modules used: node_components

subroutine: catalogprojectedcorrelationfunctionperform
Description: Compute the projected correlation function from a galaxy catalog.
Code lines: 152
Contained by: file tasks.catalog_projected_correlation_function.F90
Modules used: galacticus_display galacticus_error

galacticus_hdf5 input_parameters
io_hdf5 io_irate
iso_varying_string memory_management
numerical_constants_astronomical points
pseudo_random statistics_points_correlations
string_handling

subroutine: pointstoredshiftspace
Description: Shift the set of points into redshift space.
Code lines: 21
Contained by: subroutine catalogprojectedcorrelationfunctionperform
Modules used: vectors

interface: taskcatalogprojectedcorrelationfunction
Description: Constructors for the catalogProjectedCorrelationFunction task.
Code lines: 4
Contained by: file tasks.catalog_projected_correlation_function.F90

file: tasks.conditional_mass_function.F90
Code lines: 396
Modules used: conditional_mass_functions cosmology_functions
geometry_surveys halo_mass_functions
mass_function_incompletenesses

function: conditionalmassfunctionconstructorinternal
Description: Constructor for the conditionalMassFunction task class which takes a parameter set as input.
Code lines: 36
Contained by: file tasks.conditional_mass_function.F90
Modules used: galacticus_error memory_management
numerical_ranges

function: conditionalmassfunctionconstructorparameters
Description: Constructor for the conditionalMassFunction task class which takes a parameter set as input.
Code lines: 155
Contained by: file tasks.conditional_mass_function.F90
Modules used: galacticus_error input_parameters

subroutine: conditionalmassfunctiondestructor
Description: Destructor for the conditionalMassFunction task class.
Code lines: 11
Contained by: file tasks.conditional_mass_function.F90

subroutine: conditionalmassfunctionperform
Description: Compute and output the halo mass function.
Code lines: 133
Contained by: file tasks.conditional_mass_function.F90
Modules used: fgsl galacticus_display

galacticus_error
galacticus_hdf5
io_hdf5
iso_varying_string
memory_management
numerical_integration
string_handling

function: integrandmasshalo
Description: Integral over halo mass function.
Code lines: 9
Contained by: subroutine conditionalmassfunctionperform

function: integrandnormalizationtime
Description: Normalization integral over time.
Code lines: 7
Contained by: subroutine conditionalmassfunctionperform

function: integrandtime
Description: Integral over time.
Code lines: 11
Contained by: subroutine conditionalmassfunctionperform

interface: taskconditionalmassfunction
Description: Constructors for the conditionalMassFunction task.
Code lines: 4
Contained by: file tasks.conditional_mass_function.F90

file: tasks.evolve_forests.F90
Code lines: 1214
Modules used: galactic_filters
galacticus_nodes
input_parameters
kind_numbers
merger_tree_construction
merger_tree_operators
merger_tree_outputters
merger_trees_evolve
output_times
task_evolve_forests_work_shares
universe_operators

subroutine: evolveforestsautohook
Description: Attach to state store/restore event hooks.
Code lines: 9
Contained by: file tasks.evolve_forests.F90
Modules used: events_hooks

type: evolveforestsbranchlist
Code lines: 4
Contained by: file tasks.evolve_forests.F90

function: evolveforestsconstructorinternal
Description: Internal constructor for the evolveForests task class.
Code lines: 23
Contained by: file tasks.evolve_forests.F90

function: `evolveforestsconstructorparameters`

Description: Constructor for the `evolveForests` task class which takes a parameter set as input.
Code lines: 164
Contained by: file `tasks.evolve_forests.F90`
Modules used: `galacticus_error`, `galacticus_nodes`, `node_components`, `system_load`

subroutine: `evolveforestsdestructor`

Description: Destructor for the `evolveForests` task class.
Code lines: 21
Contained by: file `tasks.evolve_forests.F90`
Modules used: `events_hooks`, `node_components`

subroutine: `evolveforestsperform`

Description: Evolves the complete set of merger trees as specified.
Code lines: 777
Contained by: file `tasks.evolve_forests.F90`
Modules used: `events_hooks`, `galacticus_display`, `galacticus_error`, `galacticus_function_classes_destroys`, `galacticus_meta_tree_timing`, `galacticus_nodes`, `galacticus_output_halo_models`, `iso_c_binding`, `memory_management`, `merger_tree_dump_structure`, `merger_tree_output_structure`, `merger_tree_walkers`, `merger_trees_initialize`, `node_component_black_hole_standard`, `node_component_disk_standard`, `node_component_dynamics_statistics_-bars`, `node_component_inter_output_standard`, `node_component_mass_flow_statistics_-standard`, `node_component_merging_statistics_-major`, `node_component_satellite_orbiting`, `node_component_spheroid_standard`, `node_components`, `node_events_inter_tree`, `semaphores`, `sort`, `string_handling`, `system_load`

subroutine: `evolveforestsresumetree`

Description: Resume processing of a tree.
Code lines: 20
Contained by: file `tasks.evolve_forests.F90`
Modules used: `iso_varying_string`, `string_handling`

subroutine: `evolveforestsstaterestore`

Description: Store the internal state of this object.
Code lines: 16
Contained by: file `tasks.evolve_forests.F90`
Modules used: `fgsl`, `iso_c_binding`

subroutine: `evolveforestsstatestore`
Description: Store the internal state of this object.
Code lines: 16
Contained by: file tasks.evolve_forests.F90
Modules used: `fgsl` `iso_c_binding`

subroutine: `evolveforestssuspendtree`
Description: Suspend processing of a tree.
Code lines: 41
Contained by: file tasks.evolve_forests.F90
Modules used: galacticus_error iso_varying_string kind_numbers string_handling

interface: `taskevolveforests`
Description: Constructors for the `evolveForests` task.
Code lines: 4
Contained by: file tasks.evolve_forests.F90

file: `tasks.evolve_forests.utilities.F90`
Description: Contains a module of globally-accessible functions supporting the `evolveForests` task class.
Code lines: 108

module: `tasks_evolve_forests_utilities`
Description: Provides globally-accessible functions supporting the `evolveForests` task class.
Code lines: 86
Contained by: file tasks.evolve_forests.utilities.F90
Used by: subroutine functions_global_set

subroutine: `tasks_evolve_forest_construct`
Description: Build a `taskEvolveForests` object from a given parameter set. This is a globally-callable function to allow us to subvert the class/module hierarchy.
Code lines: 20
Contained by: module tasks_evolve_forests_utilities
Modules used: galacticus_error input_parameters tasks

subroutine: `tasks_evolve_forest_destruct`
Description: Destruct a `taskEvolveForests` object passed to us as an unlimited polymorphic object.
Code lines: 15
Contained by: module tasks_evolve_forests_utilities
Modules used: galacticus_error tasks

subroutine: `tasks_evolve_forest_perform`
Description: Perform the task for a `taskEvolveForests` object passed to us as an unlimited polymorphic object.
Code lines: 15
Contained by: module tasks_evolve_forests_utilities
Modules used: galacticus_error tasks

file: `tasks.evolve_forests.work_share.F90`
Description: Contains a module which provides a class that implements general tasks to be performed by Galacticus.
Code lines: 114

module: `task_evolve_forests_work_shares`
Description: Provides a class that implements general tasks to be performed by Galacticus.
Code lines: 92
Contained by: file tasks.evolve_forests.work_share.F90
Modules used: `iso_c_binding`
Used by: subroutine galacticus_function_-classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; file tasks.evolve_forests.F90

subroutine: `evolveforestsworkerids`
Description: Returns an ID for this worker which is unique across all MPI and OpenMP threads.
Code lines: 47
Contained by: module task_evolve_forests_work_shares
Modules used: mpi_utilities

file: `tasks.evolve_forests.work_share.cyclic.F90`
Code lines: 101

function: `cyclicconstructorinternal`
Description: Internal constructor for the `cyclic` forest evolution work sharing class.
Code lines: 9
Contained by: file tasks.evolve_forests.work_share.cyclic.F90

function: `cyclicconstructorparameters`
Description: Constructor for the `cyclic` forest evolution work sharing class which takes a parameter set as input.
Code lines: 11
Contained by: file tasks.evolve_forests.work_share.cyclic.F90
Modules used: input_parameters

subroutine: `cyclicdestructor`
Description: Destructor for the `cyclic` forest evolution work sharing class.
Code lines: 7
Contained by: file tasks.evolve_forests.work_share.cyclic.F90

function: `cyclicforestnumber`
Description: Return the number of the next forest to process.
Code lines: 24
Contained by: file tasks.evolve_forests.work_share.cyclic.F90
Modules used: galacticus_error

interface: `evolveforestsworksharecyclic`
Description: Constructors for the `cyclic` forest evolution work sharing class.

Code lines: 4
Contained by: file tasks.evolve_forests.work_share.cyclic.F90

file: tasks.evolve_forests.work_share.first_come_first_served.F90
Code lines: 79
Modules used: mpi_utilities

interface: evolveforestsworkshareFCFS
Description: Constructors for the fcfs forest evolution work sharing class.
Code lines: 4
Contained by: file tasks.evolve_forests.work_share.first_come_first_served.F90

function: fcfsconstructorinternal
Description: Internal constructor for the fcfs forest evolution work sharing class.
Code lines: 8
Contained by: file tasks.evolve_forests.work_share.first_come_first_served.F90

function: fcfsconstructorparameters
Description: Constructor for the fcfs forest evolution work sharing class which takes a parameter set as input.
Code lines: 11
Contained by: file tasks.evolve_forests.work_share.first_come_first_served.F90
Modules used: input_parameters

function: fcfsforestnumber
Description: Return the number of the next forest to process.
Code lines: 13
Contained by: file tasks.evolve_forests.work_share.first_come_first_served.F90
Modules used: galacticus_error

file: tasks.evolve_forests.work_share.stride.F90
Code lines: 105
Modules used: iso_c_binding

interface: evolveforestsworksharestride
Description: Constructors for the stride forest evolution work sharing class.
Code lines: 4
Contained by: file tasks.evolve_forests.work_share.stride.F90

function: strideconstructorinternal
Description: Internal constructor for the stride forest evolution work sharing class.
Code lines: 11
Contained by: file tasks.evolve_forests.work_share.stride.F90
Modules used: galacticus_error

function: strideconstructorparameters
Description: Constructor for the stride forest evolution work sharing class which takes a parameter set as input.
Code lines: 29
Contained by: file tasks.evolve_forests.work_share.stride.F90
Modules used: input_parameters

subroutine: `stridedestructor`
Description: Destructor for the `stride` forest evolution work sharing class.
Code lines: 7
Contained by: file `tasks.evolve_forests.work_share.stride.F90`

function: `strideforestnumber`
Description: Return the number of the next forest to process.
Code lines: 9
Contained by: file `tasks.evolve_forests.work_share.stride.F90`

file: `tasks.excursion_sets.F90`
Code lines: 295
Modules used: `cosmological_density_field`, `cosmology_functions`, `cosmology_parameters`, `excursion_sets_barriers`, `excursion_sets_first_crossings`, `halo_mass_functions`, `power_spectra`

function: `excursionsetsconstructorinternal`
Description: Constructor for the `excursionSets` task class which takes a parameter set as input.
Code lines: 17
Contained by: file `tasks.excursion_sets.F90`

function: `excursionsetsconstructorparameters`
Description: Constructor for the `excursionSets` task class which takes a parameter set as input.
Code lines: 110
Contained by: file `tasks.excursion_sets.F90`
Modules used: `galacticus_hdf5`, `galacticus_nodes`, `input_parameters`, `node_components`

subroutine: `excursionsetsdestructor`
Description: Destructor for the `excursionSets` task class.
Code lines: 16
Contained by: file `tasks.excursion_sets.F90`
Modules used: `node_components`

subroutine: `excursionsetsperform`
Description: Compute and output the halo mass function.
Code lines: 89
Contained by: file `tasks.excursion_sets.F90`
Modules used: `galacticus_display`, `galacticus_error`, `galacticus_hdf5`, `galacticus_nodes`, `io_hdf5`, `memory_management`, `numerical_constants_math`, `numerical_ranges`

interface: `taskexcursionsets`
Description: Constructors for the `excursionSets` task.
Code lines: 4
Contained by: file `tasks.excursion_sets.F90`

file: tasks.halo_mass_function.F90
Code lines: 462
Modules used: cosmological_density_field, cosmology_functions, cosmology_parameters, dark_matter_halo_biases, dark_matter_halo_scales, dark_matter_profile_scales, dark_matter_profiles_dmo, halo_mass_functions, linear_growth, output_times, transfer_functions, unevolved_subhalo_mass_functions, virial_density_contrast

function: haloMassFunctionConstructorInternal
Description: Constructor for the haloMassFunction task class which takes a parameter set as input.
Code lines: 25
Contained by: file tasks.halo_mass_function.F90

function: haloMassFunctionConstructorParameters
Description: Constructor for the haloMassFunction task class which takes a parameter set as input.
Code lines: 108
Contained by: file tasks.halo_mass_function.F90
Modules used: galacticus_nodes, input_parameters, node_components

subroutine: haloMassFunctionDestructor
Description: Destructor for the haloMassFunction task class.
Code lines: 24
Contained by: file tasks.halo_mass_function.F90
Modules used: node_components

subroutine: haloMassFunctionPerform
Description: Compute and output the halo mass function.
Code lines: 228
Contained by: file tasks.halo_mass_function.F90
Modules used: dark_matter_profile_scales, fgsl, galacticus_calculations_resets, galacticus_display, galacticus_error, galacticus_hdf5, galacticus_nodes, io_hdf5, iso_c_binding, memory_management, numerical_constants_astronomical, numerical_integration, numerical_ranges, string_handling

function: subhaloMassFunctionIntegrand
Description: Integrand function used to find the cumulative subhalo mass function.
Code lines: 11
Contained by: subroutine haloMassFunctionPerform

interface: taskHaloMassFunction
Description: Constructors for the haloMassFunction task.
Code lines: 4

Contained by: file tasks.halo_mass_function.F90

file: tasks.halo_model.projected_correlation_function.F90
Code lines: 284
Modules used: conditional_mass_functions, cosmology_functions, dark_matter_halo_biases, dark_matter_halo_scales, dark_matter_profile_scales, dark_matter_profiles_dmo, geometry_surveys, halo_mass_functions, linear_growth, power_spectra

function: halomodelprojectedcorrelationfunctionconstructorinternal
Description: Constructor for the haloModelProjectedCorrelationFunction task class which takes a parameter set as input.
Code lines: 26
Contained by: file tasks.halo_model.projected_correlation_function.F90
Modules used: memory_management, numerical_ranges

function: halomodelprojectedcorrelationfunctionconstructorparameters
Description: Constructor for the haloModelProjectedCorrelationFunction task class which takes a parameter set as input.
Code lines: 138
Contained by: file tasks.halo_model.projected_correlation_function.F90
Modules used: galacticus_nodes, input_parameters, node_components

subroutine: halomodelprojectedcorrelationfunctiondestructor
Description: Destructor for the haloModelProjectedCorrelationFunction task class.
Code lines: 19
Contained by: file tasks.halo_model.projected_correlation_function.F90
Modules used: node_components

subroutine: halomodelprojectedcorrelationfunctionperform
Description: Generate a mock galaxy catalog using a simple halo model approach.
Code lines: 20
Contained by: file tasks.halo_model.projected_correlation_function.F90
Modules used: galacticus_display, galacticus_error, galacticus_hdf5, halo_model_projected_correlations, io_hdf5

function: halomodelprojectedcorrelationfunctionrequiresoutputfile
Description: Specifies that this task does not requires the main output file.
Code lines: 8
Contained by: file tasks.halo_model.projected_correlation_function.F90

interface: taskhalomodelprojectedcorrelationfunction
Description: Constructors for the haloModelProjectedCorrelationFunction task.
Code lines: 4
Contained by: file tasks.halo_model.projected_correlation_function.F90

file: tasks.halo_model_generate.F90
Code lines: 361
Modules used: conditional_mass_functions, cosmology_functions, cosmology_parameters, dark_matter_profile_scales, dark_matter_profiles_dmo

function: halomodelgenerateconstructorinternal
Description: Constructor for the haloModelGenerate task class which takes a parameter set as input.
Code lines: 14
Contained by: file tasks.halo_model_generate.F90

function: halomodelgenerateconstructorparameters
Description: Constructor for the haloModelGenerate task class which takes a parameter set as input.
Code lines: 74
Contained by: file tasks.halo_model_generate.F90
Modules used: galacticus_nodes, input_parameters, node_components

subroutine: halomodelgeneratedestructor
Description: Destructor for the haloModelGenerate task class.
Code lines: 14
Contained by: file tasks.halo_model_generate.F90
Modules used: node_components

subroutine: halomodelgenerateperform
Description: Generate a mock galaxy catalog using a simple halo model approach.
Code lines: 190
Contained by: file tasks.halo_model_generate.F90
Modules used: conditional_mass_functions, galactic_structure_enclosed_masses, galactic_structure_options, galacticus_calculations_resets, galacticus_display, galacticus_error, galacticus_nodes, geometry_surveys, io_hdf5, io_irate, iso_varying_string, memory_management, numerical_constants_astronomical, numerical_constants_prefixes, pseudo_random, root_finder, string_handling

function: centralmassroot
Description: Root function used to find the mass of central galaxies
Code lines: 9
Contained by: subroutine halomodelgenerateperform

subroutine: galaxyadd
Description: Add a galaxy to the output buffers.
Code lines: 38
Contained by: subroutine halomodelgenerateperform

function: `satellitemassroot`
Description: Root function used to find the mass of satellite galaxies
Code lines: 9
Contained by: subroutine halomodelgenerateperform

function: `halomodelgenerateperform`
Description: Specifies that this task does not requires the main output file.
Code lines: 8
Contained by: file tasks.halo_model_generate.F90

interface: `taskhalomodelgenerate`
Description: Constructors for the `haloModelGenerate` task.
Code lines: 4
Contained by: file tasks.halo_model_generate.F90

file: `tasks.halo_spin_distribution.F90`
Code lines: 243
Modules used: cosmology_functions, halo_spin_distributions, output_times

function: `halospindistributionconstructorinternal`
Description: Constructor for the `haloSpinDistribution` task class which takes a parameter set as input.
Code lines: 12
Contained by: file tasks.halo_spin_distribution.F90

function: `halospindistributionconstructorparameters`
Description: Constructor for the `haloSpinDistribution` task class which takes a parameter set as input.
Code lines: 83
Contained by: file tasks.halo_spin_distribution.F90
Modules used: galacticus_nodes, input_parameters, node_components

subroutine: `halospindistributiondestructor`
Description: Destructor for the `haloSpinDistribution` task class.
Code lines: 12
Contained by: file tasks.halo_spin_distribution.F90
Modules used: node_components

subroutine: `halospindistributionperform`
Description: Compute and output the halo spin distribution.
Code lines: 82
Contained by: file tasks.halo_spin_distribution.F90
Modules used: galacticus_display, galacticus_error, galacticus_hdf5, galacticus_nodes, halo_spin_distributions, io_hdf5, iso_c_binding, string_handling

interface: `taskhalospindistribution`
Description: Constructors for the `haloSpinDistribution` task.
Code lines: 4
Contained by: file tasks.halo_spin_distribution.F90

file: `tasks.intergalactic_medium_state.F90`
Code lines: 170
Modules used: cosmology_functions, intergalactic_medium_filtering_masses, intergalactic_medium_state, output_times

function: `intergalacticmediumstateconstructorinternal`
Description: Constructor for the `intergalacticMediumState` task class which takes a parameter set as input.
Code lines: 12
Contained by: file tasks.intergalactic_medium_state.F90

function: `intergalacticmediumstateconstructorparameters`
Description: Constructor for the `intergalacticMediumState` task class which takes a parameter set as input.
Code lines: 31
Contained by: file tasks.intergalactic_medium_state.F90
Modules used: input_parameters

subroutine: `intergalacticmediumstatedestructor`
Description: Destructor for the `intergalacticMediumState` task class.
Code lines: 10
Contained by: file tasks.intergalactic_medium_state.F90

subroutine: `intergalacticmediumstateperform`
Description: Output IGM state to the Galacticus output file.
Code lines: 62
Contained by: file tasks.intergalactic_medium_state.F90
Modules used: galacticus_display, galacticus_error, galacticus_hdf5, io_hdf5, iso_c_binding, string_handling

interface: `taskintergalacticmediumstate`
Description: Constructors for the `intergalacticMediumState` task.
Code lines: 4
Contained by: file tasks.intergalactic_medium_state.F90

file: `tasks.mass_function_covariance.F90`
Code lines: 924
Modules used: conditional_mass_functions, cosmology_functions, dark_matter_halo_biases, geometry_surveys, halo_mass_functions, power_spectra_nonlinear

function: massfunctioncovarianceangularpowerintegrand
Description: Integrand for large scale structure variance computed using survey mask angular power spectrum.
Code lines: 41
Contained by: file tasks.mass_function_covariance.F90

function: massfunctioncovarianceangularpowerradialterm
Description: Computes the radial term in the expression for large scale structure variance.
Code lines: 49
Contained by: file tasks.mass_function_covariance.F90
Modules used: gamma_functions hypergeometric_functions numerical_constants_math

function: massfunctioncovariancebiasintegrandi
Description: Integral for bias.
Code lines: 9
Contained by: file tasks.mass_function_covariance.F90

subroutine: massfunctioncovariancecomputevolumenormalizations
Description: Compute volume normalization factors for LSS covariance calculations.
Code lines: 22
Contained by: file tasks.mass_function_covariance.F90
Modules used: fgsl numerical_integration

function: massfunctioncovarianceconstructorinternal
Description: Internal constructor for the massFunctionCovariance task class.
Code lines: 17
Contained by: file tasks.mass_function_covariance.F90

function: massfunctioncovarianceconstructorparameters
Description: Constructor for the powerSpectrum task class which takes a parameter set as input.
Code lines: 129
Contained by: file tasks.mass_function_covariance.F90
Modules used: input_parameters

subroutine: massfunctioncovariancedestructor
Description: Destructor for the massFunctionCovariance task class.
Code lines: 12
Contained by: file tasks.mass_function_covariance.F90

function: massfunctioncovariancegalaxyrootpowerspectrum
Description: Computes the quantity $\int_{t_{\rm min}}^{t_{\rm max}} {\rm d}t b(t) \sqrt{P(k,t)} {\rm d}V/{\rm d}t$, where $b(t)$ is galaxy bias, and $P(k,t)$ is the non-linear galaxy power spectrum.
Code lines: 17
Contained by: file tasks.mass_function_covariance.F90
Modules used: fgsl numerical_integration

function: massfunctioncovariancehalooccupancyintegrand

Description: Integral for mass function.
Code lines: 9
Contained by: file tasks.mass_function_covariance.F90

function: massfunctioncovariancehalooccupancytimeinterand
Description: Integral for comoving volume.
Code lines: 17
Contained by: file tasks.mass_function_covariance.F90
Modules used: fgsl numerical_integration

function: massfunctioncovariancelargescalestructureintegrand
Description: Integral for LSS contribution to the covariance matrix.
Code lines: 22
Contained by: file tasks.mass_function_covariance.F90
Modules used: fgsl numerical_interpolation

subroutine: massfunctioncovariancelssangularspectrum
Description: Compute variance due to large scale structure by integration over the angular power spectrum.
Code lines: 61
Contained by: file tasks.mass_function_covariance.F90
Modules used: fgsl galacticus_display
memory_management numerical_constants_math
numerical_integration

subroutine: massfunctioncovariancelsswindowfunction
Description: Compute variance due to large scale structure by directly summing over the Fourier transform of the survey selection function.
Code lines: 107
Contained by: file tasks.mass_function_covariance.F90
Modules used: fftw3 galacticus_display
iso_c_binding memory_management
numerical_constants_math

function: massfunctioncovariancemassfunctionintegrandi
Description: Integral for mass function.
Code lines: 9
Contained by: file tasks.mass_function_covariance.F90

function: massfunctioncovariancemassfunctiontimeintegrandi
Description: Integral for comoving volume.
Code lines: 17
Contained by: file tasks.mass_function_covariance.F90
Modules used: fgsl numerical_integration

subroutine: massfunctioncovarianceperform
Description: Compute and output the halo mass function.
Code lines: 263
Contained by: file tasks.mass_function_covariance.F90

Modules used: completeness, fgsl, galacticus_display, galacticus_error, io_hdf5, memory_management, number, numerical_constants_astronomical, numerical_constants_math, numerical_integration, numerical_ranges

function: massfunctioncovariancerequiresoutputfile
Description: Specifies that this task does not requires the main output file.
Code lines: 8
Contained by: file tasks.mass_function_covariance.F90

function: massfunctioncovariancevolumeintegrand
Description: Integral for comoving volume.
Code lines: 7
Contained by: file tasks.mass_function_covariance.F90

interface: taskmassfunctioncovariance
Description: Constructors for the powerSpectrum task.
Code lines: 4
Contained by: file tasks.mass_function_covariance.F90

file: tasks.merger_tree_file_builder.F90
Code lines: 567
Modules used: cosmological_density_field, cosmology_parameters, power_spectra_primordial, stateful_types, transfer_functions

function: mergertreefilebuilderconstructorinternal
Description: Constructor for the mergerTreeFileBuilder task class which takes a parameter set as input.
Code lines: 24
Contained by: file tasks.merger_tree_file_builder.F90

function: mergertreefilebuilderconstructorparameters
Description: Constructor for the mergerTreeFileBuilder task class which takes a parameter set as input.
Code lines: 353
Contained by: file tasks.merger_tree_file_builder.F90
Modules used: input_parameters, merger_tree_data_structure, numerical_constants_astronomical, numerical_constants_prefixes

subroutine: mergertreefilebuilderdestructor
Description: Destructor for the mergerTreeFileBuilder task class.
Code lines: 10
Contained by: file tasks.merger_tree_file_builder.F90

subroutine: mergertreefilebuilderperform
Description: Compute and output the halo mass function.

Code lines: 90
Contained by: file tasks.merger_tree_file_builder.F90
Modules used: dates_and_times, galacticus_display, galacticus_error, hdf5, input_parameters, merger_tree_data_structure

function: mergertreefilebuilderrequiresoutputfile
Description: Specifies that this task does not require the main output file.
Code lines: 8
Contained by: file tasks.merger_tree_file_builder.F90

type: mergertreemetadata
Description: Type used for metadata in merger tree files.
Code lines: 4
Contained by: file tasks.merger_tree_file_builder.F90

type: propertycolumn
Description: Type used to specify which property to read from which column,
Code lines: 4
Contained by: file tasks.merger_tree_file_builder.F90

interface: taskmergertreefilebuilder
Description: Constructors for the mergerTreeFileBuilder task.
Code lines: 4
Contained by: file tasks.merger_tree_file_builder.F90

file: tasks.multi.F90
Code lines: 172

function: multiconstructorinternal
Description: Internal constructor for the multi task class.
Code lines: 14
Contained by: file tasks.multi.F90

function: multiconstructorparameters
Description: Constructor for the multi task class which takes a parameter set as input.
Code lines: 22
Contained by: file tasks.multi.F90
Modules used: input_parameters

subroutine: multideepcopy
Description: Perform a deep copy for the multi task class.
Code lines: 31
Contained by: file tasks.multi.F90
Modules used: galacticus_error

subroutine: multidestructor
Description: Destructor for the multi task class.
Code lines: 16

Contained by: file tasks.multi.F90

subroutine: multiperform
Description: Perform all tasks.
Code lines: 19
Contained by: file tasks.multi.F90
Modules used: galacticus_display galacticus_error

function: multirequiresoutputfile
Description: Returns true if any sub-task requires that the output file be open.
Code lines: 13
Contained by: file tasks.multi.F90

type: multitasklist
Code lines: 3
Contained by: file tasks.multi.F90

interface: taskmulti
Description: Constructors for the multi task.
Code lines: 4
Contained by: file tasks.multi.F90

file: tasks.nBody_analyze.F90
Code lines: 143
Modules used: nbody_importers nbody_operators

function: nbodyanalyzeconstructorinternal
Description: Constructor for the nbodyAnalyze task class which takes a parameter set as input.
Code lines: 12
Contained by: file tasks.nBody_analyze.F90

function: nbodyanalyzeconstructorparameters
Description: Constructor for the nbodyAnalyze task class which takes a parameter set as input.
Code lines: 36
Contained by: file tasks.nBody_analyze.F90
Modules used: input_parameters

subroutine: nbodyanalyzedestructor
Description: Destructor for the nbodyAnalyze task class.
Code lines: 8
Contained by: file tasks.nBody_analyze.F90

subroutine: nbodyanalyzeperform
Description: Compute and output the halo mass function.
Code lines: 24
Contained by: file tasks.nBody_analyze.F90
Modules used: galacticus_display galacticus_error nbody_simulation_data

function: `nbodyanalyzerequiresoutputfile`
Description: Specifies that this task does not requires the main output file.
Code lines: 8
Contained by: file tasks.nBody_analyze.F90

interface: `tasknbodyanalyze`
Description: Constructors for the `nbodyAnalyze` task.
Code lines: 4
Contained by: file tasks.nBody_analyze.F90

file: `tasks.posterior_sample.F90`
Code lines: 117
Modules used: posterior_sampling_simulation

function: `posteriorsampleconstructorinternal`
Description: Internal constructor for the `posteriorSample` task class.
Code lines: 8
Contained by: file tasks.posterior_sample.F90

function: `posteriorsampleconstructorparameters`
Description: Constructor for the `posteriorSample` task class which takes a parameter set as input.
Code lines: 38
Contained by: file tasks.posterior_sample.F90
Modules used: galacticus_nodes input_parameters node_components

subroutine: `posteriorsampledestructor`
Description: Destructor for the `posteriorSample` task class.
Code lines: 9
Contained by: file tasks.posterior_sample.F90
Modules used: node_components

subroutine: `posteriorsampleperform`
Description: Perform the posterior sampling.
Code lines: 13
Contained by: file tasks.posterior_sample.F90
Modules used: galacticus_display galacticus_error

interface: `taskposteriorsample`
Description: Constructors for the `posteriorSample` task.
Code lines: 4
Contained by: file tasks.posterior_sample.F90

file: `tasks.power_spectrum.F90`
Code lines: 313
Modules used: cosmological_density_field cosmology_functions cosmology_parameters linear_growth

output_times power_spectra
power_spectra_nonlinear power_spectrum_window_functions

function: powerspectraconstructorinternal
Description: Internal constructor for the powerSpectrum task class.
Code lines: 19
Contained by: file tasks.power_spectrum.F90

function: powerspectraconstructorparameters
Description: Constructor for the powerSpectrum task class which takes a parameter set as input.
Code lines: 78
Contained by: file tasks.power_spectrum.F90
Modules used: input_parameters

subroutine: powerspectradestructor
Description: Destructor for the powerSpectrum task class.
Code lines: 14
Contained by: file tasks.power_spectrum.F90

subroutine: powerspectraperform
Description: Compute and output the halo mass function.
Code lines: 136
Contained by: file tasks.power_spectrum.F90
Modules used: fgsl galacticus_display
galacticus_error galacticus_hdf5
io_hdf5 iso_c_binding
memory_management numerical_constants_astronomical
numerical_integration numerical_ranges
string_handling

function: varianceintegrand
Description: Integrand function used in compute the variance in (real space) top-hat spheres from the power spectrum.
Code lines: 9
Contained by: subroutine powerspectraperform

interface: taskpowerspectra
Description: Constructors for the powerSpectrum task.
Code lines: 4
Contained by: file tasks.power_spectrum.F90

file: tasks.report.F90
Code lines: 77

function: reportparameters
Description: Constructor for the report task class which takes a parameter set as input.
Code lines: 10
Contained by: file tasks.report.F90
Modules used: input_parameters

subroutine: `reportperform`
Description: Builds the tabulation.
Code lines: 17
Contained by: file `tasks.report.F90`
Modules used: `galacticus_build`, `galacticus_display`, `galacticus_error`, `galacticus_versioning`

function: `reportrequiresoutputfile`
Description: Specifies that this task does not requires the main output file.
Code lines: 8
Contained by: file `tasks.report.F90`

interface: `taskreport`
Description: Constructors for the `report` task.
Code lines: 3
Contained by: file `tasks.report.F90`

file: `tests.DiemerKravtsov2014_concentration.F90`
Description: Contains a program which tests the Diemer and Kravtsov [2014] halo concentration algorithm.
Code lines: 115

program: `test_diemerkravtsov2014_concentration`
Description: Tests the Diemer and Kravtsov [2014] halo concentration algorithm. Values of concentration are taken from their website.
Code lines: 93
Contained by: file `tests.DiemerKravtsov2014_concentration.F90`
Modules used: `cosmological_density_field`, `cosmology_functions`, `cosmology_parameters`, `dark_matter_profiles_concentration`, `events_hooks`, `file_utilities`, `galacticus_display`, `galacticus_error`, `galacticus_function_classes_destroys`, `galacticus_nodes`, `galacticus_paths`, `input_parameters`, `iso_varying_string`, `node_components`, `power_spectra`, `system_command`, `unit_tests`

file: `tests.IO.HDF5.F90`
Code lines: 716

program: `tests_io_hdf5`
Description: Tests the HDF5 I/O module.
Code lines: 696
Contained by: file `tests.IO.HDF5.F90`
Modules used: `galacticus_display`, `hdf5`, `io_hdf5`, `iso_varying_string`, `kind_numbers`, `unit_tests`

file: `tests.IO.XML.F90`
Description: Contains a program which tests functionality of the XML I/O module.
Code lines: 96

program: `tests_io_xml`
Description: Tests the XML I/O module.
Code lines: 74
Contained by: file `tests.IO.XML.F90`
Modules used: `fox_dom`, `galacticus_display`, `galacticus_error`, `io_xml`, `system_command`, `unit_tests`

file: `tests.MPI.F90`
Description: Contains a program to test MPI functions.
Code lines: 56

program: `test_mpi`
Description: Tests of MPI functions.
Code lines: 34
Contained by: file `tests.MPI.F90`
Modules used: `galacticus_display`, `iso_c_binding`, `mpi`, `mpi_utilities`, `unit_tests`

file: `tests.NFW96_concentration.dark_energy.F90`
Description: Contains a program which tests the Navarro et al. [1996] halo concentration algorithm in a dark energy Universe. Comparisons are made to the "`charden`" code written by Julio Navarro.
Code lines: 106

program: `test_nfw96_concentration_dark_energy`
Description: Tests the Navarro et al. [1996] halo concentration algorithm in a dark energy Universe. Comparisons are made to the "`charden`" code written by Julio Navarro.
Code lines: 82
Contained by: file `tests.NFW96_concentration.dark_energy.F90`
Modules used: `cosmology_functions`, `cosmology_parameters`, `dark_matter_profiles_concentration`, `events_hooks`, `galacticus_display`, `galacticus_function_classes_destroys`, `galacticus_nodes`, `input_parameters`, `iso_varying_string`, `node_components`, `string_handling`, `unit_tests`

file: `tests.ODEIV2_solver.F90`
Description: Contains a program to test ODE-IV2 solver routines.
Code lines: 91

program: `test_ode_solver`
Description: Tests that ODE solver routines work.
Code lines: 69

Contained by: file tests.ODEIV2_solver.F90
Modules used: fodeiv2, galacticus_display, iso_c_binding, numerical_integration2, odeiv2_solver, test_ode_solver_functions, unit_tests

file: tests.ODE_solver.F90
Description: Contains a program to test ODE solver routines.
Code lines: 79

program: test_ode_solver
Description: Tests that ODE solver routines work.
Code lines: 57
Contained by: file tests.ODE_solver.F90
Modules used: fgsl, galacticus_display, ode_solver, test_ode_solver_functions, unit_tests

file: tests.ODE_solver.functions.F90
Description: Contains a module of ODEs for unit tests.
Code lines: 100

module: test_ode_solver_functions
Description: Contains ODEs for unit tests.
Code lines: 78
Contained by: file tests.ODE_solver.functions.F90
Modules used: fgsl
Used by: program test_ode_solver, program test_ode_solver

subroutine: integrands_set_2
Description: A set of integrands for unit tests.
Code lines: 13
Contained by: module test_ode_solver_functions

function: jacobian_set_1
Description: Jacobian for a set of ODEs for unit tests.
Code lines: 11
Contained by: module test_ode_solver_functions

function: jacobian_set_2
Description: Jacobian for a set of ODEs for unit tests.
Code lines: 14
Contained by: module test_ode_solver_functions

function: ode_set_1
Description: A set of ODEs for unit tests.
Code lines: 10
Contained by: module test_ode_solver_functions

function: `ode_set_2`
Description: A set of ODEs for unit tests.
Code lines: 11
Contained by: module `test_ode_solver_functions`

file: `tests.Prada2011_concentration.F90`
Description: Contains a program which tests the Prada et al. [2011] halo concentration algorithm.
Code lines: 100

program: `test_prada2011_concentration`
Description: Tests the Prada et al. [2011] halo concentration algorithm. Values of concentration were read from their Figure 12.
Code lines: 78
Contained by: file `tests.Prada2011_concentration.F90`
Modules used:
`cosmology_functions`
`cosmology_parameters`
`dark_matter_profiles_concentration`
`events_hooks`
`galacticus_display`
`galacticus_function_classes_destroys`
`galacticus_nodes`
`input_parameters`
`iso_varying_string`
`node_components`
`unit_tests`

file: `tests.Zhao2009_algorithms.EdS.F90`
Description: Contains a program which tests the Zhao et al. [2009] halo mass formation history and halo concentration algorithms in an Einstein-de Sitter Universe.
Code lines: 145

program: `test_zhao2009_flat`
Description: Tests the Zhao et al. [2009] halo mass formation history and halo concentration algorithms in an Einstein-de Sitter Universe. Comparisons are made to the "`mandc`" Note that comparison tolerances are relatively large since we have not attempted to match details (such as critical density calculation) with "`mandc`".
Code lines: 122
Contained by: file `tests.Zhao2009_algorithms.EdS.F90`
Modules used:
`cosmology_functions`
`dark_matter_halo_mass_accretion_histories`
`dark_matter_profiles_concentration`
`events_hooks`
`file_utilities`
`galacticus_display`
`galacticus_function_classes_destroys`
`galacticus_nodes`
`galacticus_paths`
`input_parameters`
`iso_varying_string`
`node_components`
`string_handling`
`unit_tests`

file: `tests.Zhao2009_algorithms.dark_energy.F90`
Description: Contains a program which tests the Zhao et al. [2009] halo mass formation history and halo concentration algorithms in a dark energy Universe.
Code lines: 146

program: `test_zhao2009_dark_energy`

Description: Tests the Zhao et al. [2009] halo mass formation history and halo concentration algorithms in a dark energy Universe. Comparisons are made to the "mandc" Note that comparison tolerances are relatively large since we have not attempted to match details (such as critical density calculation) with "mandc".
Code lines: 123
Contained by: file tests.Zhao2009_algorithms.dark_energy.F90
Modules used: cosmology_functions, dark_matter_halo_mass_accretion_histories, dark_matter_profiles_concentration, events_hooks, file_utilities, galacticus_display, galacticus_function_classes_destroys, galacticus_nodes, galacticus_paths, input_parameters, iso_varying_string, node_components, string_handling, unit_tests

file: tests.Zhao2009_algorithms.open.F90
Description: Contains a program which tests the Zhao et al. [2009] halo mass formation history and halo concentration algorithms in an open Universe.
Code lines: 146

program: test_zhao2009_open
Description: Tests the Zhao et al. [2009] halo mass formation history and halo concentration algorithms in an open Universe. Comparisons are made to the "mandc" Note that comparison tolerances are relatively large since we have not attempted to match details (such as critical density calculation) with "mandc".
Code lines: 123
Contained by: file tests.Zhao2009_algorithms.open.F90
Modules used: cosmology_functions, dark_matter_halo_mass_accretion_histories, dark_matter_profiles_concentration, events_hooks, file_utilities, galacticus_display, galacticus_function_classes_destroys, galacticus_nodes, galacticus_paths, input_parameters, iso_varying_string, node_components, string_handling, unit_tests

file: tests.abundances.F90
Description: Contains a program to test abundances objects functions.
Code lines: 56

program: test_abundances
Description: Test abundances objects.
Code lines: 34
Contained by: file tests.abundances.F90
Modules used: abundances_structure, galacticus_display, input_parameters, iso_varying_string, unit_tests

file: tests.accretion_disks.F90

Description: Contains a program to test accretion disk functions.
Code lines: 56

program: `test_accretion_disks`
Description: Tests of accretion disk functions.
Code lines: 34
Contained by: file `tests.accretion_disks.F90`
Modules used: `accretion_disks` `galacticus_display` `galacticus_nodes` `numerical_constants_physical` `numerical_constants_prefixes` `unit_tests`

file: `tests.arrays.F90`
Description: Contains a program to test the array functions.
Code lines: 197

program: `test_array_monotonicity`
Description: Tests that array functions.
Code lines: 175
Contained by: file `tests.arrays.F90`
Modules used: `array_utilities` `galacticus_display` `iso_varying_string` `kind_numbers` `unit_tests`

file: `tests.black_hole_fundamentals.F90`
Description: Contains a program to test the black hole fundamental functions.
Code lines: 51

program: `test_black_hole_fundamentals`
Description: Tests of black hole fundamental functions.
Code lines: 29
Contained by: file `tests.black_hole_fundamentals.F90`
Modules used: `black_hole_fundamentals` `galacticus_display` `unit_tests`

file: `tests.bug745815.F90`
Code lines: 98

program: `tests_bug745815`
Description: Tests for regression of Bug #745815 (http://bugs.launchpad.net/galacticus/+bug/745815): Skipping of a node during a tree walk.
Code lines: 78
Contained by: file `tests.bug745815.F90`
Modules used: `galacticus_display` `galacticus_nodes` `input_parameters` `iso_varying_string` `kind_numbers` `merger_tree_walkers` `unit_tests`

file: `tests.comoving_distance.F90`
Code lines: 98

program: `tests_comoving_distance`

Description: Tests comoving distance calculations for various universes. Distances calculated using Python implementation of Ned Wright's cosmology calculator.
Code lines: 78
Contained by: file `tests.comoving_distance.F90`
Modules used: `cosmology_functions`, `cosmology_functions_options`, `cosmology_parameters`, `galacticus_display`, `input_parameters`, `iso_varying_string`, `unit_tests`

file: `tests.comparisons.F90`

Description: Contains a program to test numerical comparison functions.
Code lines: 45

program: `test_comparison`

Description: Tests that numerical comparison functions work.
Code lines: 23
Contained by: file `tests.comparisons.F90`
Modules used: `galacticus_display`, `numerical_comparison`, `unit_tests`

file: `tests.concentration.Correa2015.F90`

Description: Contains a program which tests the Correa et al. [2015] concentration-mass relation.
Code lines: 86

program: `test_correa2015_concentration`

Description: Tests the Correa et al. [2015] concentration-mass relation.
Code lines: 64
Contained by: file `tests.concentration.Correa2015.F90`
Modules used: `cosmology_functions`, `dark_matter_profiles_concentration`, `events_hooks`, `galacticus_display`, `galacticus_function_classes_destroys`, `galacticus_nodes`, `galacticus_paths`, `input_parameters`, `iso_varying_string`, `node_components`, `unit_tests`

file: `tests.cooling_functions.F90`

Description: Contains a program which tests cooling function functionality.
Code lines: 99

program: `test_cooling_functions`

Description: Tests cooling function functionality.
Code lines: 77
Contained by: file `tests.cooling_functions.F90`
Modules used: `abundances_structure`, `chemical_abundances_structure`, `chemical_states`, `cooling_functions`, `cosmology_functions`, `galacticus_display`, `galacticus_paths`, `input_parameters`

iso_varying_string
numerical_constants_astronomical
numerical_constants_physical
numerical_constants_units
radiation_fields
unit_tests

file: tests.cosmic_age.F90
Code lines: 111

program: tests_cosmic_age
Description: Tests cosmic age calculations for various Universers. Ages calculated using Python implementation of Ned Wright's cosmology calculator.
Code lines: 91
Contained by: file tests.cosmic_age.F90
Modules used: cosmology_functions, cosmology_parameters, galacticus_display, input_parameters, iso_varying_string, numerical_constants_math, unit_tests

file: tests.crash.F90
Description: Contains a program to test that crashes are detected.
Code lines: 26

program: test_crash
Description: Tests that crashes are detected.
Code lines: 4
Contained by: file tests.crash.F90

file: tests.dark_matter_halo_radius_enclosing_mass.F90
Description: Contains a program which tests the calculation of dark matter halo radius enclosing a given mass.
Code lines: 135

program: test_dark_matter_halo_radius_enclosing_mass
Description: Tests the calculation of dark matter halo radius enclosing a given mass.
Code lines: 111
Contained by: file tests.dark_matter_halo_radius_enclosing_mass.F90
Modules used: dark_matter_halo_scales, dark_matter_profiles_dmo, dark_matter_profiles_generic, events_hooks, galacticus_display, galacticus_nodes, input_parameters, iso_varying_string, node_components, unit_tests

file: tests.dark_matter_profiles.F90
Description: Contains a program which tests dark matter profiles.
Code lines: 96

program: test_dark_matter_profiles
Description: Tests dark matter profiles.
Code lines: 74
Contained by: file tests.dark_matter_profiles.F90

Modules used: cosmology_functions, dark_matter_halo_scales, dark_matter_profiles_dmo, events_hooks, galacticus_display, galacticus_nodes, input_parameters, iso_varying_string, node_components, unit_tests

file: `tests.dark_matter_profiles.generic.F90`
Description: Contains a program to test calculations for generic dark matter profiles.
Code lines: 343

program: `test_dark_matter_profiles_generic`
Description: Tests that numerical differentiation functions work.
Code lines: 321
Contained by: file tests.dark_matter_profiles.generic.F90
Modules used: cosmology_functions, cosmology_parameters, dark_matter_halo_scales, dark_matter_profiles, dark_matter_profiles_dmo, dark_matter_profiles_generic, events_hooks, functions_global_utilities, galacticus_display, galacticus_nodes, input_parameters, node_components, unit_tests

file: `tests.dark_matter_profiles.heated.F90`
Description: Contains a program which tests the nodes implementation.
Code lines: 101

program: `test_dark_matter_profiles_heated`
Description: Tests the heated dark matter profile implementation. An isothermal dark matter halo is used, since analytic solutions are available for this case. Specifically, the initial radius in the unheated profile is given by $r_\mathrm{i}(r) = (r_\mathrm{h}^4/4r^2+r_\mathrm{h}^2)^{1/2}-r_\mathrm{h}^2/2r$ where r is the radius in the heated profile, and $r_\mathrm{h} = (\mathrm{G}M_\mathrm{v}/2Qr_\mathrm{v})^{1/2}$ is a characteristic heating radius. Here, M_v, and r_v are the virial mass and radius of the halo respectively, and Qr_i^2 is the specific heat input to the density profile, with Q assumed to be a constant (as expected for tidal heating). Assuming no shell crossing, the enclosed mass in the final profile is simply $M(r) = M_\mathrm{v}r_\mathrm{i}(r)/r_\mathrm{v}$, from which the density of the final profile is found as $\rho(r) = (4\pi r^2)^{-1}\mathrm{d}M(r)/\mathrm{d}r$.
Code lines: 79
Contained by: file tests.dark_matter_profiles.heated.F90
Modules used: dark_matter_halo_scales, dark_matter_profiles_dmo, dark_matter_profiles_generic, events_hooks, galacticus_display, galacticus_nodes, input_parameters, iso_varying_string, numerical_constants_astronomical, numerical_constants_math, unit_tests

file: `tests.differentiation.F90`
Description: Contains a program to test differentiation functions.
Code lines: 48

program: `test_differentiation`

Description: Tests that numerical differentiation functions work.
Code lines: 26
Contained by: file tests.differentiation.F90
Modules used: galacticus_display, numerical_differentiation, test_differentiation_functions, unit_tests

file: tests.differentiation.functions.F90
Description: Contains a module of functions for differentation unit tests.
Code lines: 39

module: test_differentiation_functions
Description: Contains functions for differentation unit tests.
Code lines: 17
Contained by: file tests.differentiation.functions.F90
Used by: program test_differentiation

function: function1
Description: Function for unit testing.
Code lines: 7
Contained by: module test_differentiation_functions

file: tests.fail.F90
Description: Contains a program to test that failures are detected.
Code lines: 26

program: test_fail
Description: Tests that failures are detected.
Code lines: 4
Contained by: file tests.fail.F90

file: tests.gaunt_factors.F90
Description: Contains a program which tests Gaunt factor functions.
Code lines: 57

program: test_gaunt_factors
Description: Tests Gaunt factor functions.
Code lines: 35
Contained by: file tests.gaunt_factors.F90
Modules used: atomic_ionization_potentials, atomic_radiation_gaunt_factors, galacticus_display, input_parameters, iso_varying_string, unit_tests

file: tests.geometry.coordinate_systems.F90
Description: Contains a program to coordinate system functions.
Code lines: 55

program: test_coordinate_systems
Description: Tests of coordinate system functions.
Code lines: 33
Contained by: file tests.geometry.coordinate_systems.F90

Modules used: coordinate_systems, galacticus_display, numerical_constants_math, unit_tests

file: tests.halo_mass_function.Tinker.F90
Description: Contains a program which tests the Tinker et al. [2008] mass function by comparing to Jeremy Tinker's code.
Code lines: 100

program: tests_halo_mass_function_tinker
Description: Tests the Tinker et al. [2008] mass function by comparing to Jeremy Tinker's code.
Code lines: 77
Contained by: file tests.halo_mass_function.Tinker.F90
Modules used: cosmological_density_field, cosmology_functions, cosmology_parameters, file_utilities, galacticus_display, halo_mass_functions, input_parameters, iso_varying_string, memory_management, unit_tests

file: tests.hashes.F90
Description: Contains a program to test features of the hashes (i.e. associative arrays) module.
Code lines: 77

program: test_hashes
Description: Tests features of the hashes (i.e. associative arrays) module.
Code lines: 55
Contained by: file tests.hashes.F90
Modules used: galacticus_display, hashes, input_parameters, unit_tests

file: tests.hashes.cryptographic.F90
Description: Contains a program to test features of cryptographic hashes.
Code lines: 46

program: test_hashes_cryptographic
Description: Contains a program to test features of cryptographic hashes.
Code lines: 24
Contained by: file tests.hashes.cryptographic.F90
Modules used: galacticus_display, hashes_cryptographic, iso_varying_string, unit_tests

file: tests.hashes.perfect.F90
Description: Contains a program to test perfect hashing algorithms.
Code lines: 74

program: test_perfect_hashes
Description: Tests perfect hashing algorithms.
Code lines: 52
Contained by: file tests.hashes.perfect.F90
Modules used: galacticus_display, hashes_perfect

kind_numbers memory_management
unit_tests

file: `tests.initial_mass_functions.F90`
Description: Contains a program to test stellar inital mass functions.
Code lines: 102

program: `test_initial_mass_functions`
Description: Tests of stellar initial mass functions.
Code lines: 78
Contained by: file tests.initial_mass_functions.F90
Modules used: galacticus_display, numerical_integration2, stellar_populations_initial_mass_functions, unit_tests

function: `initialmassfunctionintegrand`
Description: Integrand used to find the total mass in the initial mass function.
Code lines: 7
Contained by: program test_initial_mass_functions

file: `tests.integration.F90`
Description: Contains a program to test integration routines.
Code lines: 64

program: `test_integration`
Description: Tests that numerical integration routines work.
Code lines: 42
Contained by: file tests.integration.F90
Modules used: fgsl, galacticus_display, numerical_constants_math, numerical_integration, test_integration_functions, unit_tests

file: `tests.integration.functions.F90`
Description: Contains a module of integrands for unit tests.
Code lines: 73

module: `test_integration_functions`
Description: Contains integrands for unit tests.
Code lines: 51
Contained by: file tests.integration.functions.F90
Modules used: fgsl, galacticus_display
Used by: program test_integration

function: `integrand1`
Description: Integral for unit testing.
Code lines: 7
Contained by: module test_integration_functions

function: `integrand2`

Description: Integral for unit testing.
Code lines: 7
Contained by: module test_integration_functions

function: integrand3
Description: Integral for unit testing.
Code lines: 7
Contained by: module test_integration_functions

function: integrand4
Description: Integral for unit testing.
Code lines: 8
Contained by: module test_integration_functions
Modules used: numerical_integration

file: tests.integration2.F90
Description: Contains a program to test integration routines.
Code lines: 335

program: test_integration2
Description: Tests that numerical integration routines work.
Code lines: 313
Contained by: file tests.integration2.F90
Modules used: fgsl, galacticus_display, galacticus_error, iso_c_binding, iso_varying_string, kind_numbers, numerical_integration, numerical_integration2, test_integration2_functions, yeplibrary

file: tests.integration2.functions.F90
Description: Contains a module of integrands for unit tests.
Code lines: 238

module: test_integration2_functions
Description: Contains integrands for unit tests.
Code lines: 216
Contained by: file tests.integration2.functions.F90
Modules used: iso_varying_string, numerical_integration2
Used by: program test_integration2

subroutine: function14vector
Description: Combined functions number 1 and 4 for numerical integration tests: vector version.
Code lines: 13
Contained by: module test_integration2_functions

function: function1scalar
Description: Test function number 1 for numerical integration tests: scalar version.
Code lines: 7
Contained by: module test_integration2_functions

function: `function1vector`
Description: Test function number 1 for numerical integration tests: vector version.
Code lines: 7
Contained by: module `test_integration2_functions`

function: `function1yeppp`
Description: Test function number 1 for numerical integration tests: YEPPP! vector version.
Code lines: 17
Contained by: module `test_integration2_functions`
Modules used: `iso_c_binding` `yepcore` `yepmath`

function: `function2scalar`
Description: Test function number 1 for numerical integration tests: scalar version.
Code lines: 7
Contained by: module `test_integration2_functions`

function: `function2vector`
Description: Test function number 2 for numerical integration tests: vector version.
Code lines: 7
Contained by: module `test_integration2_functions`

function: `function2yeppp`
Description: Test function number 2 for numerical integration tests: YEPPP! vector version.
Code lines: 17
Contained by: module `test_integration2_functions`
Modules used: `iso_c_binding` `yepcore` `yepmath`

function: `function3scalar`
Description: Test function number 3 for numerical integration tests: scalar version.
Code lines: 7
Contained by: module `test_integration2_functions`

function: `function3vector`
Description: Test function number 3 for numerical integration tests: vector version.
Code lines: 8
Contained by: module `test_integration2_functions`

function: `function3yeppp`
Description: Test function number 3 for numerical integration tests: YEPPP! vector version.
Code lines: 18
Contained by: module `test_integration2_functions`
Modules used: `iso_c_binding` `yepcore` `yepmath`

function: `function4scalar`
Description: Test function number 4 for numerical integration tests: scalar version.
Code lines: 7

Contained by: module test_integration2_functions

function: `function4vector`
Description: Test function number 4 for numerical integration tests: vector version.
Code lines: 8
Contained by: module test_integration2_functions

type: `testfunction`
Description: Type used for referencing functions.
Code lines: 9
Contained by: module test_integration2_functions

type: `testfunctionmulti`
Description: Type used for referencing functions.
Code lines: 6
Contained by: module test_integration2_functions

subroutine: `testfunctionsinitialize`
Description: Initalize an array of test functions for integration tests.
Code lines: 10
Contained by: module test_integration2_functions

type: `testintegrator`
Description: Type used for testing numerical integrators.
Code lines: 6
Contained by: module test_integration2_functions

type: `testintegratormulti`
Description: Type used for testing multi-integrand numerical integrators.
Code lines: 5
Contained by: module test_integration2_functions

file: `tests.interpolation.2D.F90`
Description: Contains a program to test 2D interpolation routines.
Code lines: 61

program: `test_interpolation_2d`
Description: Tests that 2D interpolation routines work.
Code lines: 39
Contained by: file tests.interpolation.2D.F90
Modules used: galacticus_display numerical_interpolation_2d_irregular unit_tests

file: `tests.interpolation.F90`
Description: Contains a program to test the numerical interpolation code.
Code lines: 67

program: `test_interpolation`
Description: Tests that numerical interpolation code works correctly.
Code lines: 45

Contained by: file tests.interpolation.F90
Modules used: fgsl, galacticus_display, numerical_interpolation, table_labels, unit_tests

file: tests.kepler_orbits.F90
Code lines: 138

program: tests_kepler_orbits
Description: Tests for orbital parameter conversions.
Code lines: 118
Contained by: file tests.kepler_orbits.F90
Modules used: galacticus_display, input_parameters, iso_varying_string, kepler_orbits, numerical_constants_physical, unit_tests

file: tests.linear_growth.EdS.F90
Code lines: 59

program: tests_linear_growth_eds
Description: Tests linear growth calculations.
Code lines: 39
Contained by: file tests.linear_growth.EdS.F90
Modules used: cosmology_functions, galacticus_display, input_parameters, iso_varying_string, linear_growth, unit_tests

file: tests.linear_growth.cosmological_constant.F90
Code lines: 64

program: tests_linear_growth_cosmological_constant
Description: Tests linear growth calculations for a cosmological constant Universe. Growth rates are compared to calculations taken from Andrew Hamilton's "growl" code available at: http://casa.colorado.edu/ ajsh/growl/
Code lines: 44
Contained by: file tests.linear_growth.cosmological_constant.F90
Modules used: cosmology_functions, galacticus_display, input_parameters, iso_varying_string, linear_growth, memory_management, unit_tests

file: tests.linear_growth.dark_energy.F90
Code lines: 61

program: tests_linear_growth_dark_energy
Description: Tests linear growth calculations for a dark energy Universe. Growth rates are compared to Figure 1 of Linder and Jenkins (2003; MNRAS; 346; 573; http://adsabs.harvard.edu/abs/2003MNRAS.346..573L).
Code lines: 41
Contained by: file tests.linear_growth.dark_energy.F90
Modules used: cosmology_functions, galacticus_display

input_parameters iso_varying_string
linear_growth unit_tests

file: tests.linear_growth.open.F90
Code lines: 62

program: tests_linear_growth_open
Description: Tests linear growth calculations for an open Universe. Growth rates are compared to calculations taken from: http://www.icosmos.co.uk/index.html
Code lines: 42
Contained by: file tests.linear_growth.open.F90
Modules used: cosmology_functions galacticus_display
input_parameters iso_varying_string
linear_growth unit_tests

file: tests.locks.F90
Description: Contains a program to test OpenMP lock functions.
Code lines: 75

program: test_locks
Description: Tests of OpenMP locking functions.
Code lines: 53
Contained by: file tests.locks.F90
Modules used: array_utilities galacticus_display
iso_c_binding iso_varying_string
locks string_handling
unit_tests

file: tests.make_ranges.F90
Description: Contains a program to test the numerical range making code.
Code lines: 58

program: test_make_ranges
Description: Tests that numerical range making code works correctly.
Code lines: 36
Contained by: file tests.make_ranges.F90
Modules used: array_utilities galacticus_display
numerical_ranges unit_tests

file: tests.mass_accretion_history.Correa2015.F90
Description: Contains a program which tests the Correa et al. [2015] halo mass formation history.
Code lines: 83

program: test_correa2015_mah
Description: Tests the Correa et al. [2015] halo mass formation history algorithm.
Code lines: 61
Contained by: file tests.mass_accretion_history.Correa2015.F90
Modules used: cosmology_functions dark_matter_halo_mass_accretion_histories

events_hooks
galacticus_display
galacticus_function_classes_destroys
galacticus_nodes
galacticus_paths
input_parameters
iso_varying_string
node_components
unit_tests

file: tests.mass_distributions.F90
Description: Contains a program to test mass distributions.
Code lines: 132

program: test_mass_distributions
Description: Tests mass distributions.
Code lines: 110
Contained by: file tests.mass_distributions.F90
Modules used: coordinates
galacticus_display
galacticus_error
mass_distributions
numerical_constants_math
unit_tests

file: tests.math.fast.F90
Description: Contains a program to test mathematical special functions.
Code lines: 81

program: test_math_fast
Description: Tests of mathematical fast functions.
Code lines: 59
Contained by: file tests.math.fast.F90
Modules used: galacticus_display
kind_numbers
math_exponentiation
unit_tests

file: tests.math_distributions.F90
Description: Contains a program to test mathematical distributions.
Code lines: 86

program: test_math_distributions
Description: Tests of mathematical distributions.
Code lines: 64
Contained by: file tests.math_distributions.F90
Modules used: galacticus_display
input_parameters
math_distributions_poisson_binomial
pseudo_random
statistics_distributions
unit_tests

file: tests.math_special_functions.F90
Description: Contains a program to test mathematical special functions.
Code lines: 107

program: test_math_special_functions
Description: Tests of mathematical special functions.
Code lines: 85
Contained by: file tests.math_special_functions.F90

Modules used: bessel_functions, error_functions, exponential_integrals, factorials, galacticus_display, gamma_functions, hypergeometric_functions, unit_tests

file: tests.meshes.F90
Description: Contains a program to test the array functions.
Code lines: 76

program: test_meshes
Description: Test mesh functions.
Code lines: 54
Contained by: file tests.meshes.F90
Modules used: galacticus_display, iso_c_binding, meshes, unit_tests

file: tests.multi_counters.F90
Description: Contains a program to test multi-counters.
Code lines: 70

program: test_multi_counters
Description: Tests of multi-counters.
Code lines: 48
Contained by: file tests.multi_counters.F90
Modules used: galacticus_display, iso_c_binding, multi_counters, unit_tests

file: tests.nodes.F90
Description: Contains a program which tests the nodes implementation.
Code lines: 158

program: test_nodes
Description: Tests the nodes implementation.
Code lines: 136
Contained by: file tests.nodes.F90
Modules used: array_utilities, galacticus_display, galacticus_error, galacticus_nodes, input_parameters, iso_varying_string, memory_management, test_nodes_tasks, unit_tests

file: tests.nodes.task.F90
Description: Contains a module which implements a simple test of mapping a function over all components in a node.
Code lines: 90

module: test_nodes_tasks
Description: Implements a simple test of mapping a function over all components in a node.
Code lines: 68
Contained by: file tests.nodes.task.F90

Used by: program test_nodes

subroutine: `test_node_task`
Description: Implements simple tests of mapping functions over all components in a node.
Code lines: 29
Contained by: module test_nodes_tasks
Modules used: galacticus_display, galacticus_nodes, unit_tests

function: `testfuncdouble0`
Description: A simple test function which returns the enclosed mass for a component. Used in testing mapping over a function over all components.
Code lines: 11
Contained by: module test_nodes_tasks
Modules used: galactic_structure_options, galacticus_display, galacticus_nodes

subroutine: `testvoidfunc`
Description: A simple void function used in testing mapping over a function over all components.
Code lines: 11
Contained by: module test_nodes_tasks
Modules used: galacticus_display, galacticus_nodes, iso_varying_string

file: `tests.parameters.F90`
Description: Contains a program which tests parameter input.
Code lines: 70

program: `test_parameters`
Description: Test reading of input parameters.
Code lines: 48
Contained by: file tests.parameters.F90
Modules used: cosmological_density_field, cosmology_parameters, galacticus_display, input_parameters, io_hdf5, iso_c_binding, iso_varying_string, unit_tests

file: `tests.power_spectrum.F90`
Description: Contains a program that tests power spectrum calculations.
Code lines: 99

program: `tests_power_spectrum`
Description: Tests power spectrum calculations.
Code lines: 77
Contained by: file tests.power_spectrum.F90
Modules used: cosmological_density_field, cosmology_functions, cosmology_parameters, dark_matter_particles, galacticus_display, input_parameters, iso_varying_string, numerical_constants_math

power_spectra power_spectra_primordial
power_spectra_primordial_transferred transfer_functions
unit_tests

file: tests.random.F90
Description: Contains a program to test random number functions.
Code lines: 62

program: test_random
Description: Tests that random number functions work.
Code lines: 40
Contained by: file tests.random.F90
Modules used: galacticus_display input_parameters
pseudo_random unit_tests

file: tests.regular_expressions.F90
Description: Contains a program which tests regular expression functionality.
Code lines: 46

program: tests_regular_expressions
Description: Tests regular expression functionality.
Code lines: 24
Contained by: file tests.regular_expressions.F90
Modules used: galacticus_display regular_expressions
unit_tests

file: tests.root_finding.F90
Description: Contains a program to test root finding routines.
Code lines: 107

program: test_root_finding
Description: Tests that routine finding routines work.
Code lines: 85
Contained by: file tests.root_finding.F90
Modules used: galacticus_display galacticus_error
root_finder test_root_finding_functions
unit_tests

file: tests.root_finding.functions.F90
Description: Contains a module of functions for root finding unit tests.
Code lines: 107

module: test_root_finding_functions
Description: Contains functions for root finding unit tests.
Code lines: 85
Contained by: file tests.root_finding.functions.F90
Modules used: galacticus_display
Used by: program test_root_finding

function: `root_function_1`
Description: Function for root finding unit tests.
Code lines: 5
Contained by: module `test_root_finding_functions`

function: `root_function_2`
Description: Function for root finding unit tests.
Code lines: 5
Contained by: module `test_root_finding_functions`

subroutine: `root_function_2_both`
Description: Function for root finding unit tests.
Code lines: 7
Contained by: module `test_root_finding_functions`

function: `root_function_2_derivative`
Description: Function for root finding unit tests.
Code lines: 5
Contained by: module `test_root_finding_functions`

function: `root_function_3`
Description: Function for root finding unit tests.
Code lines: 5
Contained by: module `test_root_finding_functions`

function: `root_function_4`
Description: Function for root finding unit tests.
Code lines: 10
Contained by: module `test_root_finding_functions`

subroutine: `root_function_4_both`
Description: Function for root finding unit tests.
Code lines: 13
Contained by: module `test_root_finding_functions`

function: `root_function_4_derivative`
Description: Function for root finding unit tests.
Code lines: 10
Contained by: module `test_root_finding_functions`

file: `tests.search.F90`
Description: Contains a program to test array search functions.
Code lines: 71

program: `test_search`
Description: Tests that array search functions work.
Code lines: 49
Contained by: file `tests.search.F90`
Modules used: `arrays_search` `galacticus_display`

iso_varying_string kind_numbers
unit_tests

file: tests.sigma.F90
Code lines: 89

program: tests_sigma
Description: Tests
Code lines: 69
Contained by: file tests.sigma.F90
Modules used: cosmological_density_field cosmology_parameters
galacticus_display input_parameters
iso_varying_string numerical_constants_math
numerical_ranges power_spectra_primordial
power_spectra_primordial_transferred power_spectrum_window_functions
transfer_functions unit_tests

file: tests.sort.F90
Description: Contains a program to test sorting functions.
Code lines: 67

program: test_sort
Description: Tests of sorting functions.
Code lines: 45
Contained by: file tests.sort.F90
Modules used: galacticus_display iso_c_binding
kind_numbers sort
unit_tests

file: tests.sort.topological.F90
Description: Contains a program to test sorting functions.
Code lines: 65

program: test_sort_topological
Description: Tests of topological sorting functions.
Code lines: 43
Contained by: file tests.sort.topological.F90
Modules used: galacticus_display galacticus_error
sorting_topological unit_tests

file: tests.spectra.postprocess.Inoue2014.F90
Description: Contains a program to test the Inoue et al. [2014] algorithm for IGM absorption.
Code lines: 60

program: test_inoue2014
Description: Tests the Inoue et al. [2014] algorithm for IGM absorption.
Code lines: 38
Contained by: file tests.spectra.postprocess.Inoue2014.F90
Modules used: galacticus_display numerical_constants_atomic

stellar_population_spectra_postprocess unit_tests

file: tests.spherical_collapse.dark_energy.EdS.F90

Description: Contains a program which tests spherical collapse calculations for a dark energy Universe, specifically using an Einstein-de Sitter cosmology (e.g. Kitayama and Suto 1996; eqn. A2).

Code lines: 75

program: tests_spherical_collapse_dark_energy_eds

Description: Tests spherical collapse calculations for a dark energy Universe, specifically using an Einstein-de Sitter cosmology. Compares results to the analytic solution.

Code lines: 52

Contained by: file tests.spherical_collapse.dark_energy.EdS.F90

Modules used: cosmological_density_field, cosmology_functions, galacticus_display, input_parameters, iso_varying_string, numerical_constants_math, unit_tests, virial_density_contrast

file: tests.spherical_collapse.dark_energy.constantEoSminus0.6.F90

Description: Contains a program which tests spherical collapse calculations for a dark energy Universe, specifically using a flat, $\omega = -0.6$ cosmology.

Code lines: 68

program: tests_spherical_collapse_dark_energy_omega_zero_point_six

Description: Tests spherical collapse calculations for a dark energy Universe, specifically using a flat, $\omega = -0.6$ cosmology. Compares results to points read from Figure 6 of Horellou and Berge [2005] using DataThief.

Code lines: 45

Contained by: file tests.spherical_collapse.dark_energy.constantEoSminus0.6.F90

Modules used: cosmology_functions, galacticus_display, input_parameters, iso_varying_string, unit_tests, virial_density_contrast

file: tests.spherical_collapse.dark_energy.constantEoSminus0.8.F90

Description: Contains a program which tests spherical collapse calculations for a dark energy Universe, specifically using a flat, $\omega = -0.8$ cosmology.

Code lines: 68

program: tests_spherical_collapse_dark_energy_omega_zero_point_eight

Description: Tests spherical collapse calculations for a dark energy Universe, specifically using a flat, $\omega = -0.8$ cosmology. Compares results to points read from Figure 6 of Horellou and Berge [2005] using DataThief.

Code lines: 45

Contained by: file tests.spherical_collapse.dark_energy.constantEoSminus0.8.F90

Modules used: cosmology_functions, galacticus_display, input_parameters, iso_varying_string, unit_tests, virial_density_contrast

file: tests.spherical_collapse.dark_energy.constantEoSminusHalf.F90

Description: Contains a program which tests spherical collapse calculations for a dark energy Universe, specifically using a flat, $\omega = -1/2$ cosmology.
Code lines: 73

program: `tests_spherical_collapse_dark_energy_omega_half`
Description: Tests spherical collapse calculations for a dark energy Universe, specifically using a flat, $\omega = -1/2$ cosmology. Compares results to the fitting function of Weinberg and Kamionkowski (2003; eqn. 18).
Code lines: 50
Contained by: file `tests.spherical_collapse.dark_energy.constantEoSminusHalf.F90`
Modules used:
`cosmological_density_field` `cosmology_functions`
`galacticus_display` `input_parameters`
`iso_varying_string` `linear_growth`
`numerical_constants_math` `unit_tests`

file: `tests.spherical_collapse.dark_energy.constantEoSminusTwoThirds.F90`
Description: Contains a program which tests spherical collapse calculations for a dark energy Universe, specifically using a flat, $\omega = -2/3$ cosmology.
Code lines: 73

program: `tests_spherical_collapse_dark_energy_omega_two_thirds`
Description: Tests spherical collapse calculations for a dark energy Universe, specifically using a flat, $\omega = -2/3$ cosmology. Compares results to the fitting function of Weinberg and Kamionkowski (2003; eqn. 18).
Code lines: 50
Contained by: file `tests.spherical_collapse.dark_energy.constantEoSminusTwoThirds.F90`
Modules used:
`cosmological_density_field` `cosmology_functions`
`galacticus_display` `input_parameters`
`iso_varying_string` `linear_growth`
`numerical_constants_math` `unit_tests`

file: `tests.spherical_collapse.dark_energy.lambda.F90`
Description: Contains a program which tests spherical collapse calculations for a dark energy Universe, specifically using a flat, cosmological constant cosmology.
Code lines: 80

program: `tests_spherical_collapse_dark_energy_lambda`
Description: Tests spherical collapse calculations for a dark energy Universe, specifically using a flat, cosmological constant cosmology. Compares results to the fitting function of Kitayama and Suto (1996; eqn. A6).
Code lines: 57
Contained by: file `tests.spherical_collapse.dark_energy.lambda.F90`
Modules used:
`cosmological_density_field` `cosmology_functions`
`galacticus_display` `input_parameters`
`iso_varying_string` `linear_growth`
`numerical_constants_math` `unit_tests`
`virial_density_contrast`

file: `tests.spherical_collapse.dark_energy.open.F90`

Description: Contains a program which tests spherical collapse calculations for a dark energy Universe, specifically using an open cosmology.
Code lines: 71

program: `tests_spherical_collapse_dark_energy_open`
Description: Tests spherical collapse calculations for a dark energy Universe, specifically using an open cosmology. Compares results to the analytic solution (e.g. Kitayama and Suto 1996; eqn. A4).
Code lines: 48
Contained by: file `tests.spherical_collapse.dark_energy.open.F90`
Modules used:
`cosmological_density_field`
`cosmology_functions`
`galacticus_display`
`input_parameters`
`iso_varying_string`
`linear_growth`
`numerical_constants_math`
`unit_tests`

file: `tests.spherical_collapse.flat.F90`
Description: Contains a program which tests spherical collapse calculations for a flat Universe.
Code lines: 68

program: `tests_spherical_collapse_flat`
Description: Tests spherical collapse calculations for a flat Universe. Compares results to the fitting formula of Bryan and Norman [1998].
Code lines: 46
Contained by: file `tests.spherical_collapse.flat.F90`
Modules used:
`cosmology_functions`
`galacticus_display`
`input_parameters`
`iso_varying_string`
`numerical_constants_math`
`unit_tests`
`virial_density_contrast`

file: `tests.spherical_collapse.nonlinear.F90`
Description: Contains a program which tests nonlinear collapse solution in an Einstein-de Sitter cosmology.
Code lines: 60

program: `tests_spherical_collapse_nonlinear`
Description: Tests nonlinear collapse solution in an Einstein-de Sitter cosmology.
Code lines: 38
Contained by: file `tests.spherical_collapse.nonlinear.F90`
Modules used:
`cosmology_functions`
`cosmology_parameters`
`galacticus_display`
`input_parameters`
`linear_growth`
`spherical_collapse_matter_lambda`
`tables`
`unit_tests`

file: `tests.spherical_collapse.open.F90`
Description: Contains a program which tests spherical collapse calculations for an open Universe.
Code lines: 68

program: `tests_spherical_collapse_open`
Description: Tests spherical collapse calculations for an open Universe. Compares results to the fitting formula of Bryan and Norman [1998].

Code lines: 46
Contained by: file tests.spherical_collapse.open.F90
Modules used: cosmology_functions, galacticus_display, input_parameters, iso_varying_string, numerical_constants_math, unit_tests, virial_density_contrast

file: tests.stellar_populations.F90
Description: Contains a program to test stellar populations.
Code lines: 79

program: test_stellar_populations
Description: Tests of stellar populations.
Code lines: 55
Contained by: file tests.stellar_populations.F90
Modules used: abundances_structure, galacticus_display, galacticus_paths, input_parameters, iso_varying_string, numerical_constants_astronomical, stellar_astrophysics, stellar_astrophysics_tracks, stellar_astrophysics_winds, stellar_feedback, stellar_population_spectra, stellar_populations, stellar_populations_initial_mass_functions, supernovae_population_iii, supernovae_type_ia, unit_tests

file: tests.stellar_populations.luminosities.F90
Description: Contains a program to test stellar population luminosities.
Code lines: 154

program: test_stellar_populations_luminosities
Description: Tests of stellar population luminosities.
Code lines: 130
Contained by: file tests.stellar_populations.luminosities.F90
Modules used: abundances_structure, cosmology_functions, cosmology_parameters, events_hooks, file_utilities, galacticus_display, galacticus_output_open, galacticus_paths, input_parameters, instruments_filters, iso_varying_string, numerical_constants_astronomical, stellar_astrophysics, stellar_astrophysics_tracks, stellar_astrophysics_winds, stellar_feedback, stellar_population_luminosities, stellar_population_spectra, stellar_population_spectra_postprocess, stellar_populations, stellar_populations_initial_mass_functions, supernovae_population_iii, supernovae_type_ia, unit_tests

file: `tests.string_utilities.F90`
Description: Contains a program to test string handling utilities
Code lines: 91

program: `test_string_utilities`
Description: Tests that numerical range making code works correctly.
Code lines: 69
Contained by: file `tests.string_utilities.F90`
Modules used: `galacticus_display` `iso_varying_string` `kind_numbers` `string_handling` `unit_tests`

file: `tests.tables.F90`
Description: Contains a program to test tables.
Code lines: 181

program: `test_tables`
Description: Tests that tables work correctly.
Code lines: 159
Contained by: file `tests.tables.F90`
Modules used: `array_utilities` `galacticus_display` `tables` `unit_tests`

file: `tests.tensors.F90`
Description: Contains a program to test tensor functionality.
Code lines: 64

program: `test_tensors`
Description: Tests of coordinate system functions.
Code lines: 42
Contained by: file `tests.tensors.F90`
Modules used: `galacticus_display` `tensors` `unit_tests`

file: `tests.transfer_functions.F90`
Description: Contains a program that tests transfer function calculations.
Code lines: 67

program: `tests_transfer_functions`
Description: Tests transfer function calculations.
Code lines: 45
Contained by: file `tests.transfer_functions.F90`
Modules used: `cosmology_functions` `cosmology_parameters` `dark_matter_particles` `galacticus_display` `transfer_functions` `unit_tests`

file: `tests.tree_branch_destroy.F90`
Code lines: 88

program: tests_tree_branch_destroy
Code lines: 68
Contained by: file tests.tree_branch_destroy.F90
Modules used: galacticus_display, galacticus_nodes, input_parameters, kind_numbers, unit_tests

file: tests.vectors.F90
Description: Contains a program to test vector functions.
Code lines: 87

program: test_vectors
Description: Tests of vector functions.
Code lines: 65
Contained by: file tests.vectors.F90
Modules used: galacticus_display, iso_varying_string, kind_numbers, unit_tests, vectors

file: thermodynamics.ideal_gases.F90
Description: Contains a module which implements thermodynamic properties of ideal gases.
Code lines: 61

module: ideal_gases_thermodynamics
Description: Implements thermodynamic properties of ideal gases.
Code lines: 39
Contained by: file thermodynamics.ideal_gases.F90
Used by: function bondi_hoyle_lyttleton_accretion_radius, function bondi_hoyle_lyttleton_accretion_rate, subroutine node_component_black_hole_standard_mass_accretion_rate

function: ideal_gas_jeans_length
Description: Return the Jeans length (in Mpc) for gas of given temperature and density).
Code lines: 8
Contained by: module ideal_gases_thermodynamics
Modules used: numerical_constants_physical

function: ideal_gas_sound_speed
Description: Return the sound speed (in km/s) for an ideal gas of given temperature and (optionally) meanAtomicMass.
Code lines: 19
Contained by: module ideal_gases_thermodynamics
Modules used: numerical_constants_astronomical, numerical_constants_physical

file: thermodynamics.radiation.F90
Code lines: 74

module: `thermodynamics_radiation`
Description: Implements calculations of thermal radiation.
Code lines: 54
Contained by: file `thermodynamics.radiation.F90`
Used by: function `blackbodyflux`

function: `blackbody_emission`
Description: Compute the Planck blackbody spectral radiance (defined per unit wavelength, in units of J s^{-1} m^{-2} sr^{-1} $Å^{-1}$) or J s^{-1} m^{-2} sr^{-1} Hz^{-1} depending on the optional `radianceType` argument). Input `wavelength` is in Angstroms, input temperature is in Kelvin.
Code lines: 40
Contained by: module `thermodynamics_radiation`
Modules used: `numerical_constants_physical` `numerical_constants_units`

file: `tidal_stripping.mass_loss_rate.disks.F90`
Description: Contains a module which provides a class that implements tidal stripping in disks.
Code lines: 40

module: `tidal_stripping_mass_loss_rate_disks`
Description: Provides a class that implements calculations of tidal stripping in disks.
Code lines: 18
Contained by: file `tidal_stripping.mass_loss_rate.disks.F90`
Modules used: `galacticus_nodes`
Used by: subroutine `galacticus_function_-classes_destroy` subroutine `galacticus_state_retrieve`
subroutine `galacticus_state_store` module `node_component_disk_standard`

file: `tidal_stripping.mass_loss_rate.disks.null.F90`
Description: Implementation of a null tidal stripping of disks class.
Code lines: 61

function: `nullconstructorparameters`
Description: Constructor for the `null` timescale for star formation feedback in disks class which takes a parameter set as input.
Code lines: 11
Contained by: file `tidal_stripping.mass_loss_rate.disks.null.F90`
Modules used: `input_parameters`

function: `nullratemassloss`
Description: Returns a zero mass loss rate due to tidal stripping.
Code lines: 9
Contained by: file `tidal_stripping.mass_loss_rate.disks.null.F90`

interface: `tidalstrippingdisksnull`
Description: Constructors for the `null` model of tidal stripping of disks class.
Code lines: 3
Contained by: file `tidal_stripping.mass_loss_rate.disks.null.F90`

file: `tidal_stripping.mass_loss_rate.disks.simple.F90`
Description: Implementation of a simple tidal stripping of disks class.
Code lines: 148
Modules used: satellites_tidal_fields

function: `simpleconstructorinternal`
Description: Internal constructor for the `simple` model of tidal stripping of disks class.
Code lines: 9
Contained by: file tidal_stripping.mass_loss_rate.disks.simple.F90

function: `simpleconstructorparameters`
Description: Constructor for the `simple` timescale for star formation feedback in disks class which takes a parameter set as input.
Code lines: 23
Contained by: file tidal_stripping.mass_loss_rate.disks.simple.F90
Modules used: input_parameters

subroutine: `simpledestructor`
Description: Destructor for the `simple` model of tidal stripping of disks class.
Code lines: 7
Contained by: file tidal_stripping.mass_loss_rate.disks.simple.F90

function: `simpleratemassloss`
Description: Computes the mass loss rate from disks due to tidal stripping assuming a simple model. Specifically, the mass loss rate is

$$\dot{M} = -\alpha M/\tau_{\rm disk}, \tag{18.53}$$

where

$$\alpha = F_{\rm tidal}/F_{\rm gravity}, \tag{18.54}$$

$F_{\rm tidal} = \mathcal{F}_{\rm tidal} r_{1/2}$, $\mathcal{F}_{\rm tidal}$ is the tidal field from the host halo (see §12.42), and

$$F_{\rm gravity} = V_{1/2}^2(r_{1/2})/r_{1/2} \tag{18.55}$$

is the gravitational restoring force in the disk at the half-mass radius, $r_{1/2}$.
Code lines: 58
Contained by: file tidal_stripping.mass_loss_rate.disks.simple.F90
Modules used: galactic_structure_options, galactic_structure_rotation_curves, galacticus_nodes, numerical_constants_astronomical, numerical_constants_math, numerical_constants_physical

interface: `tidalstrippingdiskssimple`
Description: Constructors for the `simple` model of tidal stripping of disks class.
Code lines: 4
Contained by: file tidal_stripping.mass_loss_rate.disks.simple.F90

file: `tidal_stripping.mass_loss_rate.spheroids.F90`
Description: Contains a module which provides a class that implements tidal stripping in spheroids.

Code lines: 40

module: `tidal_stripping_mass_loss_rate_spheroids`
Description: Provides a class that implements calculations of tidal stripping in spheroids.
Code lines: 18
Contained by: file tidal_stripping.mass_loss_rate.spheroids.F90
Modules used: galacticus_nodes
Used by: subroutine galacticus_function_-classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; module node_component_spheroid_-standard

file: `tidal_stripping.mass_loss_rate.spheroids.null.F90`
Description: Implementation of a null tidal stripping of spheroids class.
Code lines: 61

function: `nullconstructorparameters`
Description: Constructor for the `null` timescale for star formation feedback in spheroids class which takes a parameter set as input.
Code lines: 11
Contained by: file tidal_stripping.mass_loss_rate.spheroids.null.F90
Modules used: input_parameters

function: `nullratemassloss`
Description: Returns a zero mass loss rate due to tidal stripping.
Code lines: 9
Contained by: file tidal_stripping.mass_loss_rate.spheroids.null.F90

interface: `tidalstrippingspheroidsnull`
Description: Constructors for the `null` model of tidal stripping of spheroids class.
Code lines: 3
Contained by: file tidal_stripping.mass_loss_rate.spheroids.null.F90

file: `tidal_stripping.mass_loss_rate.spheroids.simple.F90`
Description: Implementation of a simple tidal stripping of spheroids class.
Code lines: 148
Modules used: satellites_tidal_fields

function: `simpleconstructorinternal`
Description: Internal constructor for the `simple` model of tidal stripping of spheroids class.
Code lines: 9
Contained by: file tidal_stripping.mass_loss_rate.spheroids.simple.F90

function: `simpleconstructorparameters`
Description: Constructor for the `simple` timescale for star formation feedback in spheroids class which takes a parameter set as input.
Code lines: 23
Contained by: file tidal_stripping.mass_loss_rate.spheroids.simple.F90
Modules used: input_parameters

subroutine: `simpledestructor`
Description: Destructor for the `simple` model of tidal stripping of spheroids class.
Code lines: 7
Contained by: file tidal_stripping.mass_loss_rate.spheroids.simple.F90

function: `simpleratemassloss`
Description: Computes the mass loss rate from spheroids due to tidal stripping assuming a simple model. Specifically, the mass loss rate is

$$\dot{M} = -\alpha M/\tau_{\rm spheroid}, \tag{18.56}$$

where

$$\alpha = F_{\rm tidal}/F_{\rm gravity}, \tag{18.57}$$

$F_{\rm tidal} = \mathcal{F}_{\rm tidal} r_{1/2}$, $\mathcal{F}_{\rm tidal}$ is the tidal field from the host halo (see §12.42), and

$$F_{\rm gravity} = V_{1/2}^2(r_{1/2})/r_{1/2} \tag{18.58}$$

is the gravitational restoring force in the spheroid at the half-mass radius, $r_{1/2}$.
Code lines: 58
Contained by: file tidal_stripping.mass_loss_rate.spheroids.simple.F90
Modules used: galactic_structure_options; galactic_structure_rotation_curves; galacticus_nodes; numerical_constants_astronomical; numerical_constants_math; numerical_constants_physical

interface: `tidalstrippingspheroidssimple`
Description: Constructors for the `simple` model of tidal stripping of spheroids class.
Code lines: 4
Contained by: file tidal_stripping.mass_loss_rate.spheroids.simple.F90

file: `universe.operators.F90`
Description: Contains a module which provides a class that implements operators on universes.
Code lines: 41

module: `universe_operators`
Description: Provides a class that implements operators on universes.
Code lines: 19
Contained by: file universe.operators.F90
Modules used: galacticus_nodes
Used by: subroutine galacticus_function_classes_destroy; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; file tasks.evolve_forests.F90

file: `universe.operators.identity.F90`
Code lines: 58

function: `identityconstructorparameters`
Description: Constructor for the `identity` universe operator class which takes a parameter set as input.
Code lines: 10

Contained by: file universe.operators.identity.F90
Modules used: input_parameters

subroutine: identityoperate
Description: Perform an identity operation on a universe.
Code lines: 8
Contained by: file universe.operators.identity.F90

interface: universeoperatoridentity
Description: Constructors for the identity universe operator.
Code lines: 3
Contained by: file universe.operators.identity.F90

file: universe.operators.intergalactic_medium_state_evolve.F90
Code lines: 732
Modules used: atomic_cross_sections_ionization_photo, atomic_ionization_potentials, atomic_radiation_gaunt_factors, atomic_rates_excitation_collisional, atomic_rates_ionization_collisional, atomic_rates_recombination_dielectronic, atomic_rates_recombination_radiative, cosmological_density_field, cosmology_functions, cosmology_parameters, galacticus_nodes, intergalactic_medium_state, linear_growth, output_times, radiation_fields

function: intergalacticmediumstateevolveconstructorinternal
Description: Internal constructor for the intergalacticMediumStateEvolve universeOperator class.
Code lines: 122
Contained by: file universe.operators.intergalactic_medium_state_evolve.F90
Modules used: galacticus_error, intergalactic_medium_filtering_masses, iso_c_binding, memory_management, numerical_comparison, numerical_constants_astronomical, numerical_constants_atomic, numerical_constants_math, numerical_constants_units, numerical_ranges

function: intergalacticmediumstateevolveconstructorparameters
Description: Constructor for the intergalacticMediumStateEvolve universeOperator class which takes a parameter set as input.
Code lines: 86
Contained by: file universe.operators.intergalactic_medium_state_evolve.F90
Modules used: input_parameters

subroutine: intergalacticmediumstateevolvedestructor
Description: Destructor for the intergalacticMediumStateEvolve universeOperator class.
Code lines: 20
Contained by: file universe.operators.intergalactic_medium_state_evolve.F90

function: `intergalacticmediumstateevolveodes`
Description: Evaluates the ODEs controlling the evolution temperature.
Code lines: 210
Contained by: file universe.operators.intergalactic_medium_state_evolve.F90
Modules used: fgsl, intergalactic_medium_filtering_masses, numerical_constants_astronomical, numerical_constants_atomic, numerical_constants_math, numerical_constants_physical, numerical_constants_prefixes, numerical_constants_units, numerical_integration, ode_solver_error_codes

function: `integrandphotoionizationheatingrate`
Description: Integrand function used to compute the rate of photoionization heating of an ionic species.
Code lines: 14
Contained by: function intergalacticmediumstateevolveodes

function: `integrandphotoionizationrate`
Description: Integrand function used to compute the rate of photoionizations of an ionic species.
Code lines: 14
Contained by: function intergalacticmediumstateevolveodes

subroutine: `intergalacticmediumstateevolveoperate`
Description: Attach an initial event to a merger tree to cause the properties update function to be called.
Code lines: 14
Contained by: file universe.operators.intergalactic_medium_state_evolve.F90
Modules used: galacticus_nodes

subroutine: `intergalacticmediumstateevolvestateset`
Code lines: 23
Contained by: file universe.operators.intergalactic_medium_state_evolve.F90
Modules used: iso_c_binding

function: `intergalacticmediumstateevolveupdate`
Description: Update the properties for a given universe.
Code lines: 153
Contained by: file universe.operators.intergalactic_medium_state_evolve.F90
Modules used: arrays_search, fodeiv2, galactic_structure_options, galacticus_display, galacticus_error, galacticus_hdf5, galacticus_nodes, io_hdf5, iso_c_binding, iso_varying_string, numerical_constants_astronomical, numerical_constants_math, numerical_constants_physical, numerical_constants_prefixes, numerical_constants_units, numerical_integration, odeiv2_solver

interface: `universeoperatorintergalacticmediumstateevolve`
Description: Constructors for the `intergalacticMediumStateEvolve` universeOperator.

Code lines: 4
Contained by: file universe.operators.intergalactic_medium_state_evolve.F90

file: utility.IO.HDF5.F90
Description: Contains a module that implements simple and convenient interfaces to a variety of HDF5 functionality.
Code lines: 14262

module: io_hdf5
Description: Implements simple and convenient interfaces to a variety of HDF5 functionality.
Code lines: 14237
Contained by: file utility.IO.HDF5.F90
Modules used: h5tb, hdf5, iso_c_binding, iso_varying_string, locks
Used by: program xray_absorption_ism_wilms2000; subroutine hopkins2007buildfile; subroutine fileloadfile; subroutine ciefilereadfile; subroutine ciefilereadfile; module galacticus_hdf5; subroutine localgroupmassfunctionfinalize; function blackholebulgerelationconstructorinternal; function colordistributionsdssconstructorinternal; function concentrationdistributioncdmcococonstructorinter; function concentrationvshalomasscdmludlow2016constructorinternal; function correlationfunctionconstructorfile; subroutine correlationfunctionfinalize; function disksizeinclinationconstructorinternal; function galaxysizessdssconstructorinternal; function luminosityfunctionconstructorfile; function luminosityfunctionhalphaconstructorfile; function massfunctionhiconstructorfile; function massfunctionstellarconstructorfile; function massmetallicityandrews2013constructorinternal; function massmetallicityblanc2017constructorinternal; subroutine meanfunction1dfinalize; function morphologicalfractiongamamoffett2016constructorinternal; subroutine morphologicalfractiongamamoffett2016finalize; subroutine scatterfunction1dfinalize; function spindistributionbett2007constructorinternal; function stellarvshalomassrelationleauthaud2012constructorinternal; subroutine volumefunction1dfinalize; subroutine galacticus_output_halo_-model_initialize; subroutine galacticus_version_output; function geometrymangleangularpower; function geometrymanglesolidangle; subroutine caputi2011ukidssudsrandomsinitialize; function mangleangularpower

subroutine emissionlinedatabaseinitialize
subroutine filter_response_load
subroutine interface_camb_transfer_function
subroutine interface_cloudy_cie_tabulate
subroutine interface_fsps_ssps_tabulate
function recfastconstructorinternal
subroutine filereaddata
subroutine readhdf5read
file merger_trees.construct.read.importer.SussingMergerTreesHDF5.F90
file merger_trees.construct.read.importer.galacticus.F90
subroutine augmentfinalize
subroutine conditionalmffinalize
subroutine informationcontentfinalize
file merger_trees.operators.mass_accretion_history.F90
subroutine outputrootmassesfinalize
subroutine particulateoperate
subroutine profilerfinalize
module merger_tree_output_structure
file merger_trees.outputter.standard.F90
subroutine standarddumpdoublebuffer
subroutine standarddumpintegerbuffer
subroutine standardfinalize
subroutine standardoutput
subroutine standardoutputgroupcreate
subroutine merger_trees_render_dump
function halomassfunctionconstructorinternal
function massfunctionconstructorinternal
function projectedcorrelationfunctionconstructorinternal
function spindistributionconstructorinternal
file nBody.import.GadgetBinary.F90
file nBody.import.GadgetHDF5.F90
subroutine rotationcurveoperate
subroutine velocitydispersionoperate
function lmnstyemssnlineconstructorinternal
module nbody_simulation_data
subroutine merger_tree_data_structure_export_galacticus
subroutine merger_tree_data_structure_export_irate
subroutine store_unit_attributes_galacticus
subroutine store_unit_attributes_irate
module galacticus_nodes
subroutine node_component_dark_matter_profile_scale_tree_output
subroutine node_component_dark_matter_profile_scale_shape_tree_output
subroutine node_component_dynamics_statistics_bars_output
function intergalacticbackgroundinternalupdate
subroutine satellite_merging_output
subroutine insituoutput
subroutine metallicitysplitoutput
function fileconstructorinternal
subroutine stellar_population_luminosity_tabulate
subroutine fspsreadfile
subroutine filereadfile
function standardinterpolate
subroutine farahifileread
subroutine farahifilewrite
subroutine takahashi2011lensingdistributionconstruct
subroutine restore_table
subroutine store_table
subroutine filereadfile
subroutine catalogprojectedcorrelationfunctionperform
subroutine conditionalmassfunctionperform

subroutine excursionsetsperform	subroutine halomassfunctionperform
subroutine halomodelprojectedcorrelationfunctionperform	subroutine halomodelgenerateperform
subroutine halospindistributionperform	subroutine intergalacticmediumstateperform
subroutine massfunctioncovarianceperform	subroutine powerspectraperform
program tests_io_hdf5	program test_parameters
function intergalacticmediumstateevolveupdate	subroutine iratecopycosmology
subroutine iratecopysimulation	subroutine iratereadhalos
subroutine iratereadsimulation	subroutine iratewritehalos
subroutine openmp_critical_wait_times	module input_parameters

type: `hdf5object`
Description: A structure that holds properties of HDF5 objects.
Code lines: 330
Contained by: module io_hdf5

subroutine: `io_hdf5_assert_attribute_type`
Description: Asserts that an attribute is of a certain type and rank.
Code lines: 75
Contained by: module io_hdf5
Modules used: galacticus_error

subroutine: `io_hdf5_assert_dataset_type`
Description: Asserts that an dataset is of a certain type and rank.
Code lines: 61
Contained by: module io_hdf5
Modules used: galacticus_error

subroutine: `io_hdf5_assert_in_critical`
Description: Assert that we are in an `HDF5_Access` OpenMP critical block.
Code lines: 7
Contained by: module io_hdf5
Modules used: galacticus_error

function: `io_hdf5_character_types`
Description: Return datatypes for character data of a given length. Types are for Fortran native and C native types.
Code lines: 30
Contained by: module io_hdf5
Modules used: galacticus_error

subroutine: `io_hdf5_close`
Description: Close an HDF5 object.
Code lines: 101
Contained by: module io_hdf5
Modules used: galacticus_display galacticus_error

string_handling

subroutine: `io_hdf5_copy`
Description: Copy the named object to the target object.
Code lines: 16
Contained by: module io_hdf5
Modules used: galacticus_error

subroutine: `io_hdf5_create_reference_scalar_to_1d`
Description: Create a scalar reference to the 1-D `toDataset` in the HDF5 group `fromGroup`.
Code lines: 111
Contained by: module io_hdf5
Modules used: galacticus_error

subroutine: `io_hdf5_create_reference_scalar_to_2d`
Description: Create a scalar reference to the 2-D `toDataset` in the HDF5 group `fromGroup`.
Code lines: 111
Contained by: module io_hdf5
Modules used: galacticus_error

subroutine: `io_hdf5_create_reference_scalar_to_3d`
Description: Create a scalar reference to the 3-D `toDataset` in the HDF5 group `fromGroup`.
Code lines: 111
Contained by: module io_hdf5
Modules used: galacticus_error

subroutine: `io_hdf5_create_reference_scalar_to_4d`
Description: Create a scalar reference to the 4-D `toDataset` in the HDF5 group `fromGroup`.
Code lines: 111
Contained by: module io_hdf5
Modules used: galacticus_error

subroutine: `io_hdf5_create_reference_scalar_to_5d`
Description: Create a scalar reference to the 5-D `toDataset` in the HDF5 group `fromGroup`.
Code lines: 111
Contained by: module io_hdf5
Modules used: galacticus_error

function: `io_hdf5_dataset_rank`
Description: Return the rank of dataset `datasetObject`.
Code lines: 39
Contained by: module io_hdf5
Modules used: galacticus_error

function: `io_hdf5_dataset_size`
Description: Return the size of the $\texttt{dim}^{\text{th}}$ dimension of dataset `datasetObject`.
Code lines: 63
Contained by: module io_hdf5
Modules used: galacticus_error

function: `io_hdf5_datasets`
Description: Return a list of all datasets present within `thisObject`.
Code lines: 67
Contained by: module io_hdf5
Modules used: galacticus_error string_handling

subroutine: `io_hdf5_deep_copy`
Description: Make a deep copy of the object, with a new HDF5 object identifier.
Code lines: 37
Contained by: module io_hdf5
Modules used: galacticus_error

subroutine: `io_hdf5_destroy`
Description: Destroy an HDF5 object by destroying its associated varying string objects.
Code lines: 8
Contained by: module io_hdf5

subroutine: `io_hdf5_end_critical`
Description: Record that we have left an `HDF5_Access` OpenMP critical block.
Code lines: 6
Contained by: module io_hdf5

subroutine: `io_hdf5_flush`
Description: Flush an HDF5 file to disk.
Code lines: 26
Contained by: module io_hdf5
Modules used: galacticus_display galacticus_error

function: `io_hdf5_has_attribute`
Description: Check if `thisObject` has an attribute with the given `attributeName`.
Code lines: 25
Contained by: module io_hdf5
Modules used: galacticus_error

function: `io_hdf5_has_dataset`
Description: Check if `thisObject` has a dataset with the given `datasetName`.
Code lines: 40
Contained by: module io_hdf5
Modules used: galacticus_error

function: `io_hdf5_has_group`
Description: Check if `thisObject` has a group with the given `groupName`.
Code lines: 32
Contained by: module io_hdf5
Modules used: galacticus_error

subroutine: `io_hdf5_initialize`
Description: Initialize the HDF5 subsystem.

Code lines: 28
Contained by: module io_hdf5
Modules used: galacticus_error

function: io_hdf5_is_hdf5
Description: Return true if the named file is an HDF5 file.
Code lines: 15
Contained by: module io_hdf5
Modules used: file_utilities galacticus_error

function: io_hdf5_is_open
Description: Returns true if thisObject is open.
Code lines: 7
Contained by: module io_hdf5

function: io_hdf5_is_reference
Description: Return true if the input dataset is a scalar reference.
Code lines: 37
Contained by: module io_hdf5
Modules used: galacticus_error

function: io_hdf5_name
Description: Returns the path to thisObject.
Code lines: 8
Contained by: module io_hdf5

function: io_hdf5_object_type
Description: Returns the object type for thisObject.
Code lines: 7
Contained by: module io_hdf5

function: io_hdf5_open_attribute
Description: Open an attribute in inObject.
Code lines: 116
Contained by: module io_hdf5
Modules used: galacticus_error

function: io_hdf5_open_dataset
Description: Open an dataset in inObject.
Code lines: 249
Contained by: module io_hdf5
Modules used: galacticus_error

subroutine: io_hdf5_open_file
Description: Open a file and return an appropriate HDF5 object. The file name can be provided as an input parameter or, if not provided, will be taken from the stored object name in fileObject.
Code lines: 154
Contained by: module io_hdf5
Modules used: file_utilities galacticus_error

function: `io_hdf5_open_group`
Description: Open an HDF5 group and return an appropriate HDF5 object. The group name can be provided as an input parameter or, if not provided, will be taken from the stored object name in `groupObject`. The location at which to open the group is taken from either `inObject` or `inPath`.
Code lines: 115
Contained by: module `io_hdf5`
Modules used: `galacticus_error`

function: `io_hdf5_parent`
Description: Return the parent object.
Code lines: 8
Contained by: module `io_hdf5`

function: `io_hdf5_path_to`
Description: Returns the path to `thisObject`.
Code lines: 8
Contained by: module `io_hdf5`

subroutine: `io_hdf5_read_attribute_character_1d_array_allocatable`
Description: Open and read an character scalar attribute in `thisObject`.
Code lines: 107
Contained by: module `io_hdf5`
Modules used: `galacticus_error` `memory_management`

subroutine: `io_hdf5_read_attribute_character_1d_array_static`
Description: Open and read an character scalar attribute in `thisObject`.
Code lines: 108
Contained by: module `io_hdf5`
Modules used: `galacticus_error`

subroutine: `io_hdf5_read_attribute_character_scalar`
Description: Open and read an character scalar attribute in `thisObject`.
Code lines: 127
Contained by: module `io_hdf5`
Modules used: `galacticus_error`

subroutine: `io_hdf5_read_attribute_double_1d_array_allocatable`
Description: Open and read an double scalar attribute in `thisObject`.
Code lines: 92
Contained by: module `io_hdf5`
Modules used: `galacticus_error` `memory_management`

subroutine: `io_hdf5_read_attribute_double_1d_array_static`
Description: Open and read an double scalar attribute in `thisObject`.
Code lines: 93
Contained by: module `io_hdf5`
Modules used: `galacticus_error`

subroutine: `io_hdf5_read_attribute_double_scalar`
Description: Open and read an double scalar attribute in `thisObject`.
Code lines: 112
Contained by: module `io_hdf5`
Modules used: `galacticus_error`

subroutine: `io_hdf5_read_attribute_integer8_1d_array_allocatable`
Description: Open and read an integer scalar attribute in `thisObject`.
Code lines: 95
Contained by: module `io_hdf5`
Modules used: `galacticus_error` `kind_numbers` `memory_management`

subroutine: `io_hdf5_read_attribute_integer8_1d_array_static`
Description: Open and read an integer scalar attribute in `thisObject`.
Code lines: 102
Contained by: module `io_hdf5`
Modules used: `galacticus_error` `kind_numbers` `memory_management`

subroutine: `io_hdf5_read_attribute_integer8_scalar`
Description: Open and read a long integer scalar attribute in `thisObject`.
Code lines: 113
Contained by: module `io_hdf5`
Modules used: `galacticus_error` `kind_numbers`

subroutine: `io_hdf5_read_attribute_integer_1d_array_allocatable`
Description: Open and read an integer scalar attribute in `thisObject`.
Code lines: 92
Contained by: module `io_hdf5`
Modules used: `galacticus_error` `memory_management`

subroutine: `io_hdf5_read_attribute_integer_1d_array_static`
Description: Open and read an integer scalar attribute in `thisObject`.
Code lines: 93
Contained by: module `io_hdf5`
Modules used: `galacticus_error`

subroutine: `io_hdf5_read_attribute_integer_scalar`
Description: Open and read an integer scalar attribute in `thisObject`.
Code lines: 110
Contained by: module `io_hdf5`
Modules used: `galacticus_error`

subroutine: `io_hdf5_read_attribute_varstring_1d_array_allocatable`
Description: Open and read an varying string 1-D array attribute in `thisObject`.
Code lines: 84
Contained by: module `io_hdf5`

Modules used: galacticus_error

subroutine: io_hdf5_read_attribute_varstring_1d_array_allocatable_do_read
Description: Open and read an varying string 1-D array attribute in thisObject by creating a suitably-sized character variable into which it can be read.
Code lines: 21
Contained by: module io_hdf5
Modules used: memory_management

subroutine: io_hdf5_read_attribute_varstring_1d_array_static
Description: Open and read an varying string 1-D array attribute in thisObject.
Code lines: 84
Contained by: module io_hdf5
Modules used: galacticus_error

subroutine: io_hdf5_read_attribute_varstring_1d_array_static_do_read
Description: Open and read an varying string 1-D array attribute in thisObject by creating a suitably-sized character variable into which it can be read.
Code lines: 17
Contained by: module io_hdf5

subroutine: io_hdf5_read_attribute_varstring_scalar
Description: Open and read an varying string scalar attribute in thisObject.
Code lines: 85
Contained by: module io_hdf5
Modules used: galacticus_error

subroutine: io_hdf5_read_attribute_varstring_scalar_do_read
Description: Open and read an varying string scalar attribute in thisObject by creating a suitably-sized character variable into which it can be read.
Code lines: 18
Contained by: module io_hdf5

subroutine: io_hdf5_read_dataset_character_1d_array_allocatable
Description: Open and read an integer scalar dataset in thisObject.
Code lines: 295
Contained by: module io_hdf5
Modules used: galacticus_error memory_management

subroutine: io_hdf5_read_dataset_character_1d_array_static
Description: Open and read a character scalar dataset in thisObject.
Code lines: 296
Contained by: module io_hdf5
Modules used: galacticus_error

subroutine: io_hdf5_read_dataset_double_1d_array_allocatable
Description: Open and read a double scalar dataset in thisObject.
Code lines: 362
Contained by: module io_hdf5

Modules used: galacticus_error memory_management

subroutine: io_hdf5_read_dataset_double_1d_array_static
Description: Open and read a double scalar dataset in thisObject.
Code lines: 356
Contained by: module io_hdf5
Modules used: galacticus_error

subroutine: io_hdf5_read_dataset_double_2d_array_allocatable
Description: Open and read a double 2-D array dataset in thisObject.
Code lines: 361
Contained by: module io_hdf5
Modules used: galacticus_error memory_management

subroutine: io_hdf5_read_dataset_double_2d_array_static
Description: Open and read a double scalar dataset in thisObject.
Code lines: 362
Contained by: module io_hdf5
Modules used: galacticus_error

subroutine: io_hdf5_read_dataset_double_3d_array_allocatable
Description: Open and read a double 3-D array dataset in thisObject.
Code lines: 281
Contained by: module io_hdf5
Modules used: galacticus_error memory_management

subroutine: io_hdf5_read_dataset_double_3d_array_static
Description: Open and read a double scalar dataset in thisObject.
Code lines: 282
Contained by: module io_hdf5
Modules used: galacticus_error

subroutine: io_hdf5_read_dataset_double_4d_array_allocatable
Description: Open and read a double 4-D array dataset in thisObject.
Code lines: 281
Contained by: module io_hdf5
Modules used: galacticus_error memory_management

subroutine: io_hdf5_read_dataset_double_4d_array_static
Description: Open and read a double scalar dataset in thisObject.
Code lines: 282
Contained by: module io_hdf5
Modules used: galacticus_error

subroutine: io_hdf5_read_dataset_double_5d_array_allocatable
Description: Open and read a double 5-D array dataset in thisObject.
Code lines: 280
Contained by: module io_hdf5
Modules used: galacticus_error memory_management

subroutine: `io_hdf5_read_dataset_double_5d_array_static`
Description: Open and read a double scalar dataset in `thisObject`.
Code lines: 282
Contained by: module `io_hdf5`
Modules used: `galacticus_error`

subroutine: `io_hdf5_read_dataset_double_6d_array_allocatable`
Description: Open and read a double 5-D array dataset in `thisObject`.
Code lines: 280
Contained by: module `io_hdf5`
Modules used: `galacticus_error` `memory_management`

subroutine: `io_hdf5_read_dataset_double_6d_array_static`
Description: Open and read a double scalar dataset in `thisObject`.
Code lines: 282
Contained by: module `io_hdf5`
Modules used: `galacticus_error`

subroutine: `io_hdf5_read_dataset_integer8_1d_array_allocatable`
Description: Open and read a long integer scalar dataset in `thisObject`.
Code lines: 358
Contained by: module `io_hdf5`
Modules used: `galacticus_error` `kind_numbers` `memory_management`

subroutine: `io_hdf5_read_dataset_integer8_1d_array_static`
Description: Open and read a long integer scalar dataset in `thisObject`.
Code lines: 364
Contained by: module `io_hdf5`
Modules used: `galacticus_error` `kind_numbers` `memory_management`

subroutine: `io_hdf5_read_dataset_integer8_2d_array_allocatable`
Description: Open and read a double 2-D array dataset in `thisObject`.
Code lines: 363
Contained by: module `io_hdf5`
Modules used: `galacticus_error` `kind_numbers` `memory_management`

subroutine: `io_hdf5_read_dataset_integer8_2d_array_static`
Description: Open and read a double scalar dataset in `thisObject`.
Code lines: 369
Contained by: module `io_hdf5`
Modules used: `galacticus_error` `kind_numbers` `memory_management`

subroutine: `io_hdf5_read_dataset_integer_1d_array_allocatable`

Description: Open and read an integer scalar dataset in `thisObject`.
Code lines: 281
Contained by: module io_hdf5
Modules used: galacticus_error memory_management

subroutine: `io_hdf5_read_dataset_integer_1d_array_static`
Description: Open and read an integer scalar dataset in `thisObject`.
Code lines: 282
Contained by: module io_hdf5
Modules used: galacticus_error

subroutine: `io_hdf5_read_dataset_integer_2d_array_allocatable`
Description: Open and read an integer scalar dataset in `thisObject`.
Code lines: 281
Contained by: module io_hdf5
Modules used: galacticus_error memory_management

subroutine: `io_hdf5_read_dataset_integer_2d_array_static`
Description: Open and read an integer scalar dataset in `thisObject`.
Code lines: 282
Contained by: module io_hdf5
Modules used: galacticus_error

subroutine: `io_hdf5_read_dataset_varstring_1d_array_allocatable`
Description: Open and read an varying string 1-D array dataset in `thisObject`.
Code lines: 84
Contained by: module io_hdf5
Modules used: galacticus_error

subroutine: `io_hdf5_read_dataset_varstring_1d_array_allocatable_do_read`
Description: Open and read an varying string 1-D array dataset in `thisObject` by creating a suitably-sized character variable into which it can be read.
Code lines: 21
Contained by: module io_hdf5
Modules used: memory_management

subroutine: `io_hdf5_read_dataset_varstring_1d_array_static`
Description: Open and read an varying string 1-D array dataset in `thisObject`.
Code lines: 84
Contained by: module io_hdf5
Modules used: galacticus_error

subroutine: `io_hdf5_read_dataset_varstring_1d_array_static_do_read`
Description: Open and read an varying string 1-D array dataset in `thisObject` by creating a suitably-sized character variable into which it can be read.
Code lines: 17
Contained by: module io_hdf5

subroutine: `io_hdf5_read_table_character_1d_array_allocatable`

Description: Open and read a real 1D array from a table `thisObject`.
Code lines: 81
Contained by: module io_hdf5
Modules used: galacticus_error memory_management

subroutine: `io_hdf5_read_table_integer8_1d_array_allocatable`
Description: Open and read a real scalar from a table `thisObject`.
Code lines: 79
Contained by: module io_hdf5
Modules used: galacticus_error kind_numbers memory_management

subroutine: `io_hdf5_read_table_integer_1d_array_allocatable`
Description: Open and read an integer 1D array from a table `thisObject`.
Code lines: 76
Contained by: module io_hdf5
Modules used: galacticus_error memory_management

subroutine: `io_hdf5_read_table_real_1d_array_allocatable`
Description: Open and read a real 1D array from a table `thisObject`.
Code lines: 78
Contained by: module io_hdf5
Modules used: galacticus_error memory_management

subroutine: `io_hdf5_set_defaults`
Description: Sets the compression level and chunk size for dataset output.
Code lines: 19
Contained by: module io_hdf5
Modules used: galacticus_error

subroutine: `io_hdf5_start_critical`
Description: Record that we have entered an `HDF5_Access` OpenMP critical block.
Code lines: 6
Contained by: module io_hdf5

subroutine: `io_hdf5_uninitialize`
Description: Uninitialize the HDF5 subsystem.
Code lines: 19
Contained by: module io_hdf5
Modules used: galacticus_error

subroutine: `io_hdf5_write_attribute_character_1d`
Description: Open and write an character 1-D array attribute in `thisObject`.
Code lines: 90
Contained by: module io_hdf5
Modules used: galacticus_error

subroutine: `io_hdf5_write_attribute_character_scalar`
Description: Open and write an character scalar attribute in `thisObject`.

Code lines: 96
Contained by: module io_hdf5
Modules used: galacticus_error

subroutine: io_hdf5_write_attribute_double_1d
Description: Open and write an double 1-D array attribute in thisObject.
Code lines: 77
Contained by: module io_hdf5
Modules used: galacticus_error

subroutine: io_hdf5_write_attribute_double_2d
Description: Open and write an double 2-D array attribute in thisObject.
Code lines: 77
Contained by: module io_hdf5
Modules used: galacticus_error

subroutine: io_hdf5_write_attribute_double_scalar
Description: Open and write an double scalar attribute in thisObject.
Code lines: 76
Contained by: module io_hdf5
Modules used: galacticus_error

subroutine: io_hdf5_write_attribute_integer8_1d
Description: Open and write an integer 1-D array attribute in thisObject.
Code lines: 87
Contained by: module io_hdf5
Modules used: galacticus_error kind_numbers memory_management

subroutine: io_hdf5_write_attribute_integer8_scalar
Description: Open and write a long integer scalar attribute in thisObject.
Code lines: 78
Contained by: module io_hdf5
Modules used: galacticus_error kind_numbers

subroutine: io_hdf5_write_attribute_integer_1d
Description: Open and write an integer 1-D array attribute in thisObject.
Code lines: 77
Contained by: module io_hdf5
Modules used: galacticus_error

subroutine: io_hdf5_write_attribute_integer_scalar
Description: Open and write an integer scalar attribute in thisObject.
Code lines: 74
Contained by: module io_hdf5
Modules used: galacticus_error

subroutine: io_hdf5_write_attribute_logical_scalar
Description: Open and write a logical scalar attribute in thisObject.

Code lines: 13
Contained by: module io_hdf5

subroutine: io_hdf5_write_attribute_varstring_1d
Description: Open and write a varying string 1-D array attribute in thisObject.
Code lines: 12
Contained by: module io_hdf5
Modules used: string_handling

subroutine: io_hdf5_write_attribute_varstring_scalar
Description: Open and write a varying string scalar attribute in thisObject.
Code lines: 11
Contained by: module io_hdf5

subroutine: io_hdf5_write_dataset_character_1d
Description: Open and write a character 1-D array dataset in thisObject.
Code lines: 179
Contained by: module io_hdf5
Modules used: galacticus_error

subroutine: io_hdf5_write_dataset_double_1d
Description: Open and write a double 1-D array dataset in thisObject.
Code lines: 160
Contained by: module io_hdf5
Modules used: galacticus_error

subroutine: io_hdf5_write_dataset_double_2d
Description: Open and write a double 2-D array dataset in thisObject.
Code lines: 172
Contained by: module io_hdf5
Modules used: galacticus_error

subroutine: io_hdf5_write_dataset_double_3d
Description: Open and write a double 3-D array dataset in thisObject.
Code lines: 172
Contained by: module io_hdf5
Modules used: galacticus_error

subroutine: io_hdf5_write_dataset_double_4d
Description: Open and write a double 4-D array dataset in thisObject.
Code lines: 172
Contained by: module io_hdf5
Modules used: galacticus_error

subroutine: io_hdf5_write_dataset_double_5d
Description: Open and write a double 5-D array dataset in thisObject.
Code lines: 172
Contained by: module io_hdf5
Modules used: galacticus_error

subroutine: `io_hdf5_write_dataset_double_6d`
Description: Open and write a double 6-D array dataset in `thisObject`.
Code lines: 172
Contained by: module io_hdf5
Modules used: galacticus_error

subroutine: `io_hdf5_write_dataset_integer8_1d`
Description: Open and write a long integer 1-D array dataset in `thisObject`.
Code lines: 168
Contained by: module io_hdf5
Modules used: galacticus_error kind_numbers memory_management

subroutine: `io_hdf5_write_dataset_integer8_2d`
Description: Open and write a long integer 2-D array dataset in `thisObject`.
Code lines: 168
Contained by: module io_hdf5
Modules used: galacticus_error kind_numbers memory_management

subroutine: `io_hdf5_write_dataset_integer_1d`
Description: Open and write an integer 1-D array dataset in `thisObject`.
Code lines: 160
Contained by: module io_hdf5
Modules used: galacticus_error

subroutine: `io_hdf5_write_dataset_integer_2d`
Description: Open and write an integer 2-D array dataset in `thisObject`.
Code lines: 160
Contained by: module io_hdf5
Modules used: galacticus_error

subroutine: `io_hdf5_write_dataset_varstring_1d`
Description: Open and write a varying string 1-D array dataset in `thisObject`.
Code lines: 16
Contained by: module io_hdf5
Modules used: string_handling

subroutine: `io_hdf_assert_is_initialized`
Description: Check if this module has been initialized.
Code lines: 11
Contained by: module io_hdf5
Modules used: galacticus_error

file: `utility.IO.IRATE.F90`
Description: Contains a module which provides IO in IRATE format.
Code lines: 246

module: `io_irate`
Description: Provides IO in IRATE format.
Code lines: 224
Contained by: file `utility.IO.IRATE.F90`
Modules used: `cosmology_functions` `cosmology_parameters` `iso_varying_string`
Used by: subroutine subroutine `halomodelgenerateperform` `catalogprojectedcorrelationfunctionperform`

interface: `irate`
Code lines: 2
Contained by: module `io_irate`

function: `irateconstructor`
Description: Constructor for IRATE file interface class.
Code lines: 13
Contained by: module `io_irate`
Modules used: `iso_varying_string`

subroutine: `iratecopycosmology`
Description: Copy "`Cosmology`" group from one IRATE file to another.
Code lines: 15
Contained by: module `io_irate`
Modules used: `io_hdf5` `iso_varying_string`

subroutine: `iratecopysimulation`
Description: Copy "`SimulationProperties`" group from one IRATE file to another.
Code lines: 15
Contained by: module `io_irate`
Modules used: `io_hdf5` `iso_varying_string`

subroutine: `iratereadhalos`
Description: Read requested properties of halos from an IRATE file.
Code lines: 51
Contained by: module `io_irate`
Modules used: `cosmology_parameters` `io_hdf5` `iso_varying_string` `numerical_constants_astronomical` `numerical_constants_prefixes`

subroutine: `iratereadsimulation`
Description: Read requested properties of the simulation from an IRATE file.
Code lines: 15
Contained by: module `io_irate`
Modules used: `io_hdf5` `iso_varying_string`

subroutine: `iratewritehalos`
Description: Read requested properties of halos from an IRATE file.
Code lines: 43

Contained by: module io_irate
Modules used: io_hdf5 iso_varying_string numerical_constants_astronomical numerical_constants_prefixes

file: utility.IO.XML.F90
Description: Contains a module which implements various utility functions for extracting data from XML files.
Code lines: 487

module: io_xml
Description: Implements various utility functions for extracting data from XML files.
Code lines: 465
Contained by: file utility.IO.XML.F90
Modules used: fox_dom iso_varying_string
Used by: subroutine atomic_data_initialize; subroutine galacticus_version_output; subroutine filter_response_load; function localgroupdbconstructorinternal; subroutine readxmlread; function fileconstructorinternal; function; subroutine fileread; intergalacticbackgroundfileconstructorinternal; function; function; hegerwoosley2002constructorinternal; nagashima2005constructorinternal; function; function; barkana2001wdmconstructorinternal; rodriguezpuebla2016constructorinternal; function tinker2008constructorinternal; program tests_io_xml; module input_parameters; function inputparametersconstructorfilechar

type: xincludenode
Description: Type used while resolving XInclude references during XML parsing.
Code lines: 4
Contained by: module io_xml

type: xincludenodelist
Description: Type used while resolving XInclude references during XML parsing.
Code lines: 3
Contained by: module io_xml

function: xml_array_length
Description: Return the length of an array of XML elements.
Code lines: 10
Contained by: module io_xml

interface: xml_array_read
Code lines: 4
Contained by: module io_xml

subroutine: xml_array_read_one_column
Description: Read one column of data from an array of XML elements.

Code lines: 20
Contained by: module io_xml
Modules used: memory_management

interface: `xml_array_read_static`
Code lines: 5
Contained by: module io_xml

subroutine: `xml_array_read_static_one_column`
Description: Read one column of data from an array of XML elements.
Code lines: 18
Contained by: module io_xml

subroutine: `xml_array_read_two_column`
Description: Read two columns of data from an array of XML elements.
Code lines: 22
Contained by: module io_xml
Modules used: memory_management

function: `xml_extract_text`
Description: Extract the text from an XML element and return as a variable length string.
Code lines: 8
Contained by: module io_xml

subroutine: `xml_extrapolation_element_decode`
Description: Extracts information from a standard XML `extrapolationElement`. Optionally a set of `allowedMethods` can be specified—if the extracted method does not match one of these an error is issued.
Code lines: 32
Contained by: module io_xml
Modules used: galacticus_error table_labels

function: `xml_get_first_element_by_tag_name`
Description: Return a pointer to the first node in an XML node that matches the given `tagName`.
Code lines: 47
Contained by: module io_xml
Modules used: galacticus_error

subroutine: `xml_list_array_read_one_column`
Description: Read one column of data from an array of XML elements.
Code lines: 19
Contained by: module io_xml
Modules used: memory_management

subroutine: `xml_list_character_array_read_static_one_column`
Description: Read one column of character data from an array of XML elements.
Code lines: 17
Contained by: module io_xml

subroutine: `xml_list_double_array_read_static_one_column`
Description: Read one column of integer data from an array of XML elements.
Code lines: 17
Contained by: module io_xml

subroutine: `xml_list_integer_array_read_static_one_column`
Description: Read one column of integer data from an array of XML elements.
Code lines: 17
Contained by: module io_xml

function: `xml_parse`
Description: Parse an XML document, automatically resolve XInclude references.
Code lines: 137
Contained by: module io_xml
Modules used: file_utilities galacticus_error

function: `xml_path_exists`
Description: Return true if the supplied `path` exists in the supplied `xmlElement`.
Code lines: 41
Contained by: module io_xml

file: `utility.MPI.F90`
Description: Contains a module that implements useful MPI utilities.
Code lines: 1529

module: `mpi_utilities`
Description: Implements useful MPI utilities.
Code lines: 1507
Contained by: file utility.MPI.F90
Modules used: galacticus_error iso_c_binding iso_varying_string mpi_f08
Used by: program galacticus; subroutine galacticus_banner_show; subroutine galacticus_output_open_file; subroutine correlationfunctionfinalizeanalysis; subroutine volumefunction1dfinalizeanalysis; subroutine galacticus_state_retrieve; subroutine galacticus_state_store; file merger_trees.construct.read.F90; function galaxypopulationevaluate; function gaussianregressionevaluate; function halomassfunctionevaluate; function independentlikelihoodssequantialevaluate; function spindistributionevaluate; function gelmanrubinconstructorinternal; subroutine gelmanrubindestructor; function gelmanrubinisconverged; subroutine gelmanrubinlogreport; function likelihoodthresholdisconverged; function adaptivegamma; function adaptiveexponent; function adaptivesample; function annealeddifferentialevolutionacceptproposal

subroutine annealeddifferentialevolutionupdate	function differentialevolutionacceptproposal
function differentialevolutionchainselect	function differentialevolutionconstructorparameters
subroutine differentialevolutionposterior	subroutine differentialevolutionsimulate
subroutine differentialevolutionupdate	function particleswarmconstructorparameters
subroutine particleswarmposterior	subroutine particleswarmsimulate
function stochasticdifferentialevolutionacceptproposal	function tempereddifferentialevolutionacceptproposal
subroutine tempereddifferentialevolutionupdate	function correlationconstructorinternal
subroutine correlationcorrelationlengthcompute	subroutine correlationrestore
function historyconstructorinternal	subroutine historyrestore
subroutine latinhypercubeinitialize	subroutine resumeinitialize
function simpleconstructorinternal	subroutine simplerestore
function farahiprobability	subroutine farahiratetabulate
function farahimidpointprobability	subroutine farahimidpointratetabulate
subroutine evolveforestsworkerids	file tasks.evolve_forests.work_-share.first_come_first_served.F90
program test_mpi	function file_name_temporary

function: `counterconstructor`
Description: Constructor for MPI counter class.
Code lines: 30
Contained by: module mpi_utilities
Modules used: galacticus_error iso_c_binding

subroutine: `counterdestructor`
Description: Destructor for the MPI counter class.
Code lines: 12
Contained by: module mpi_utilities

function: `counterget`
Description: Return the current value of an MPI counter.
Code lines: 25
Contained by: module mpi_utilities
Modules used: galacticus_error

function: `counterincrement`
Description: Increment an MPI counter.
Code lines: 27
Contained by: module mpi_utilities
Modules used: galacticus_error

function: `mpialllogicalscalar`
Description: Return true if all of the given booleans are true over all processes.

Code lines: 26
Contained by: module mpi_utilities
Modules used: galacticus_error

function: mpianylogicalscalar
Description: Return true if any of the given booleans is true over all processes.
Code lines: 26
Contained by: module mpi_utilities
Modules used: galacticus_error

function: mpiaveragearray
Description: Average an array over all processes, returning it to all processes.
Code lines: 30
Contained by: module mpi_utilities

function: mpiaveragescalar
Description: Find the maximum values of a scalar over all processes, returning it to all processes.
Code lines: 19
Contained by: module mpi_utilities

subroutine: mpibarrier
Description: Block until all MPI processes are synchronized.
Code lines: 10
Contained by: module mpi_utilities

interface: mpicounter
Code lines: 2
Contained by: module mpi_utilities

subroutine: mpifinalize
Description: Finalize MPI.
Code lines: 13
Contained by: module mpi_utilities
Modules used: galacticus_error

function: mpigather1d
Description: Gather a 1-D array from all processes, returning it as a 2-D array.
Code lines: 15
Contained by: module mpi_utilities

function: mpigather2d
Description: Gather a 1-D array from all processes, returning it as a 2-D array.
Code lines: 15
Contained by: module mpi_utilities

function: mpigatherint1d
Description: Gather an integer 1-D array from all processes, returning it as a 2-D array.
Code lines: 15
Contained by: module mpi_utilities

function: `mpigatherintscalar`
Description: Gather an integre scalar from all processes, returning it as a 1-D array.
Code lines: 19
Contained by: module `mpi_utilities`

function: `mpigatherscalar`
Description: Gather a scalar from all processes, returning it as a 1-D array.
Code lines: 19
Contained by: module `mpi_utilities`

function: `mpigetcount`
Description: Return MPI count.
Code lines: 13
Contained by: module `mpi_utilities`

function: `mpigethostaffinity`
Description: Return host affinity of this MPI process.
Code lines: 14
Contained by: module `mpi_utilities`

function: `mpigetnodeaffinity`
Description: Return node affinity of given MPI process.
Code lines: 19
Contained by: module `mpi_utilities`

function: `mpigetnodecount`
Description: Return count of nodes used by MPI.
Code lines: 13
Contained by: module `mpi_utilities`

function: `mpigetrank`
Description: Return MPI rank.
Code lines: 13
Contained by: module `mpi_utilities`

function: `mpigetranklabel`
Description: Return MPI rank label.
Code lines: 23
Contained by: module `mpi_utilities`
Modules used: `iso_varying_string`

subroutine: `mpiinitialize`
Description: Initialize MPI.
Code lines: 64
Contained by: module `mpi_utilities`
Modules used: `galacticus_error` `hashes` `memory_management`

function: `mpiisactive`
Description: Return true if MPI is active.
Code lines: 8
Contained by: module `mpi_utilities`

function: `mpiismaster`
Description: Return true if this is the master process.
Code lines: 12
Contained by: module `mpi_utilities`

function: `mpimaxloc`
Description: Find the rank of the process having maximum values of an array over all processes, returning it to all processes.
Code lines: 28
Contained by: module `mpi_utilities`

function: `mpimaxvalarray`
Description: Find the maximum values of an array over all processes, returning it to all processes.
Code lines: 26
Contained by: module `mpi_utilities`

function: `mpimaxvalscalar`
Description: Find the maximum values of a scalar over all processes, returning it to all processes.
Code lines: 19
Contained by: module `mpi_utilities`

function: `mpimedianarray`
Description: Find the median of an array over all processes, returning it to all processes.
Code lines: 50
Contained by: module `mpi_utilities`
Modules used: `sort`

function: `mpimessagewaiting`
Description: Return true if an MPI message (matching the optional `from` and `tag` if given) is waiting for receipt.
Code lines: 23
Contained by: module `mpi_utilities`

function: `mpiminloc`
Description: Find the rank of the process having minimum values of an array over all processes, returning it to all processes.
Code lines: 28
Contained by: module `mpi_utilities`

function: `mpiminvalarray`
Description: Find the minimum values of an array over all processes, returning it to all processes.
Code lines: 26
Contained by: module `mpi_utilities`

function: `mpiminvalintarray`
Description: Find the minimum values of an array over all processes, returning it to all processes.
Code lines: 26
Contained by: module `mpi_utilities`

function: `mpiminvalintscalar`
Description: Find the minimum values of a scalar over all processes, returning it to all processes.
Code lines: 19
Contained by: module `mpi_utilities`

function: `mpiminvalscalar`
Description: Find the minimum values of a scalar over all processes, returning it to all processes.
Code lines: 19
Contained by: module `mpi_utilities`

type: `mpiobject`
Code lines: 159
Contained by: module `mpi_utilities`

function: `mpirequestdata1d`
Description: Request and receive data from other MPI processes.
Code lines: 76
Contained by: module `mpi_utilities`

function: `mpirequestdata2d`
Description: Request and receive data from other MPI processes.
Code lines: 76
Contained by: module `mpi_utilities`

function: `mpirequestdataint1d`
Description: Request and receive data from other MPI processes.
Code lines: 76
Contained by: module `mpi_utilities`

function: `mpirequestdatalogical1d`
Description: Request and receive data from other MPI processes.
Code lines: 76
Contained by: module `mpi_utilities`

function: `mpisumarraydouble`
Description: Sum an integer array over all processes, returning it to all processes.
Code lines: 28
Contained by: module `mpi_utilities`

function: `mpisumarrayint`
Description: Sum an integer array over all processes, returning it to all processes.
Code lines: 29
Contained by: module `mpi_utilities`

Modules used: galacticus_error

function: mpisumarraythreedouble
Description: Sum an rank-3 double array over all processes, returning it to all processes.
Code lines: 28
Contained by: module mpi_utilities

function: mpisumarraytwodouble
Description: Sum an rank-2 double array over all processes, returning it to all processes.
Code lines: 28
Contained by: module mpi_utilities

function: mpisumscalardouble
Description: Sum an integer scalar over all processes, returning it to all processes.
Code lines: 19
Contained by: module mpi_utilities

function: mpisumscalarint
Description: Sum an integer scalar over all processes, returning it to all processes.
Code lines: 19
Contained by: module mpi_utilities

file: utility.OpenMP.F90
Description: Contains a module that implements useful OpenMP utilities.
Code lines: 64

module: openmp_utilities
Description: Implements useful OpenMP utilities.
Code lines: 42
Contained by: file utility.OpenMP.F90
Used by: subroutine galacticus_output_close_file

subroutine: openmp_critical_wait_times
Description: Outputs collected data on OpenMP critical section wait times.
Code lines: 30
Contained by: module openmp_utilities
Modules used: galacticus_hdf5, io_hdf5, iso_varying_string, openmp_utilities_data

file: utility.OpenMP.data.F90
Description: Contains a module that implements useful OpenMP utilities.
Code lines: 41

module: openmp_utilities_data
Description: Implements data for useful OpenMP utilities.
Code lines: 19
Contained by: file utility.OpenMP.data.F90
Modules used: iso_varying_string
Used by: subroutine openmp_critical_wait_times

file: `utility.arrays.F90`
Description: Contains a module which implements useful operations on arrays.
Code lines: 437

module: `array_utilities`
Description: Contains routines which implement useful operations on arrays.
Code lines: 415
Contained by: file `utility.arrays.F90`
Used by:

function `fileconstructorinternal`	function `betaprofileconstructorinternal`
function `isothermalconstructorinternal`	function `simpleconstructorinternal`
function `whitefrenk1991constructorinternal`	function `nocoolingsatellitesconstructorinternal`
function `simplescalingconstructorinternal`	function `velocitymaximumscalingconstructorinternal`
function `bett2007constructorinternal`	function `einastoconstructorinternal`
function `enzohydrostaticconstructorinternal`	function `patejloeb2015constructorinternal`
function `ricotti2000constructorinternal`	function `betaprofileconstructorparameters`
function `readconstruct`	subroutine `sussingasciiload`
subroutine `sussingtreeindicesread`	subroutine `sussinghdf5load`
subroutine `prunelightconevalidate`	subroutine `merger_tree_data_structure_-export_irate`
subroutine `stellar_luminosities_-initialize`	function `table1d_is_monotonic`
subroutine `table_1d_reverse`	subroutine `table_logarithmic_1d_reverse`
function `listconstructorparameters`	function `intergalacticbackgroundfileconstructorinternal`
function `cole2000constructorparameters`	function `simpleconstructorparameters`
function `standardconstructorparameters`	function `dynamicaltimeconstructorinternal`
function `dynamicaltimeconstructorinternal`	function `piecewisepowerlawconstructorinternal`
function `wittgordon2003constructorinternal`	subroutine `spherical_collapse_matter_-lambda_nonlinear_mapping`
subroutine `filereadfile`	program `test_array_monotonicity`
program `test_locks`	program `test_make_ranges`
program `test_nodes`	program `test_tables`

interface: `array_cumulate`
Description: Interface to generic routines which cumulate values in an array.
Code lines: 3
Contained by: module `array_utilities`

function: `array_cumulate_double`
Description: Cumulates values in a double precision array.
Code lines: 14

Contained by: module array_utilities

interface: array_index
Description: Interface to generic routines which return a subset of an array given indices into the array.
Code lines: 6
Contained by: module array_utilities

function: array_index_double
Description: Return a subset of a double precision array given a set of indices into the array.
Code lines: 12
Contained by: module array_utilities

function: array_index_double_2d
Description: Return a subset of a 2D double precision array given a set of indices into the array.
Code lines: 28
Contained by: module array_utilities
Modules used: galacticus_error

function: array_index_integer
Description: Return a subset of an integer array given a set of indices into the array.
Code lines: 12
Contained by: module array_utilities

function: array_index_integer8
Description: Return a subset of an integer array given a set of indices into the array.
Code lines: 13
Contained by: module array_utilities
Modules used: kind_numbers

function: array_intersection_varying_string
Code lines: 22
Contained by: module array_utilities
Modules used: iso_varying_string

interface: array_is_monotonic
Description: Interface to generic routines which check if an array is monotonic.
Code lines: 4
Contained by: module array_utilities

function: array_is_monotonic_double
Description: Checks if a double precision array is monotonic.
Code lines: 82
Contained by: module array_utilities

function: array_is_monotonic_integer8
Description: Checks if an integer array is monotonic.
Code lines: 83
Contained by: module array_utilities
Modules used: kind_numbers

function: `array_is_uniform`
Description: Return true if an array is uniformly distributed (optionally in the logarithm of its values) to the given tolerance.
Code lines: 29
Contained by: module `array_utilities`
Modules used: `numerical_comparison`

interface: `array_reverse`
Description: Interface to generic routines which reverse the direction of an array.
Code lines: 5
Contained by: module `array_utilities`

function: `array_reverse_double`
Description: Reverses the direction of a double precision array.
Code lines: 11
Contained by: module `array_utilities`

function: `array_reverse_real`
Description: Reverses the direction of a real array.
Code lines: 11
Contained by: module `array_utilities`

function: `array_reverse_sizet`
Description: Reverses the direction of a real array.
Code lines: 12
Contained by: module `array_utilities`
Modules used: `iso_c_binding`

subroutine: `array_which`
Description: Return an array of indices for which `mask` is true.
Code lines: 18
Contained by: module `array_utilities`
Modules used: `galacticus_error`

interface: `operator(.intersection.)`
Code lines: 2
Contained by: module `array_utilities`

file: `utility.command_arguments.F90`
Description: Contains a module which provides an interface to read command line arguments of arbitrary type.
Code lines: 126

module: `command_arguments`
Description: Provides an interface to read command line arguments of arbitrary type.
Code lines: 103
Contained by: file `utility.command_arguments.F90`

interface: `get_argument`
Description: Generic interface to routines that read command line arguments.
Code lines: 8
Contained by: module `command_arguments`

subroutine: `get_argument_character`
Description: Reads a character command line argument.
Code lines: 8
Contained by: module `command_arguments`

subroutine: `get_argument_double`
Description: Reads a double command line argument.
Code lines: 10
Contained by: module `command_arguments`

subroutine: `get_argument_integer`
Description: Reads a integer command line argument.
Code lines: 10
Contained by: module `command_arguments`

subroutine: `get_argument_logical`
Description: Reads a logical command line argument.
Code lines: 10
Contained by: module `command_arguments`

subroutine: `get_argument_real`
Description: Reads a real command line argument.
Code lines: 10
Contained by: module `command_arguments`

subroutine: `get_argument_varying_string`
Description: Reads a varying string command line argument.
Code lines: 12
Contained by: module `command_arguments`
Modules used: `iso_varying_string`

subroutine: `get_temporary_string`
Description: Reads a command line argument into a temporary string of the correct length, and returns it as a varying string.
Code lines: 11
Contained by: module `command_arguments`
Modules used: `iso_varying_string`

file: `utility.date_and_time.F90`
Description: Contains a module which implements computation of formatted dates and times.
Code lines: 89

module: `dates_and_times`

Description: Implements computation of formatted dates and times.
Code lines: 67
Contained by: file utility.date_and_time.F90
Used by: program xray_absorption_ism_wilms2000; subroutine hopkins2007buildfile; subroutine galacticus_version_output; subroutine interface_fsps_ssps_tabulate; function recfastconstructorinternal; subroutine exportoperate; function gaussianregressionevaluate; function standardinterpolate; subroutine mergertreefilebuilderperform

function: formatted_date_and_time
Description: Return a formatted date and time.
Code lines: 49
Contained by: module dates_and_times
Modules used: iso_varying_string; string_handling

file: utility.files.F90
Description: Contains a module which implements various file-related utilities.
Code lines: 372

module: file_utilities
Description: Implements various file-related utilities.
Code lines: 347
Contained by: file utility.files.F90
Modules used: iso_c_binding; iso_varying_string
Used by: file accretion_disks.spectra.Hopkins2007.F90; subroutine ciefilereadfile; subroutine ciefilereadfile; function disksizeinclinationconstructorinternal; subroutine galacticus_build_output; subroutine galacticus_version_output; function geometrymangleangularpower; subroutine geometrymanglebuild; function geometrymanglesolidangle; subroutine bernardi2013sdssmanglefiles; subroutine caputi2011ukidssudsrandomsinitialize; subroutine liwhite2009sdssrandomsinitialize; subroutine martin2010alfalfarandomsinitialize; function mangleangularpower; subroutine filter_response_load; module interfaces_camb; subroutine interface_camb_initialize; subroutine interface_camb_transfer_function; subroutine interface_cloudy_initialize; module interfaces_cloudy_cie; subroutine interface_fsps_initialize; subroutine interface_fsps_ssps_tabulate; subroutine interface_recfast_initialize; file intergalactic_medium.state.RecFast.F90; function recfastconstructorinternal; function fileconstructorinternal; subroutine filereaddata; subroutine sussingasciiload; subroutine sussingasciiopen; subroutine sussinghdf5load; subroutine sussinghdf5open; subroutine exportoperate

function outputrootmassesconstructorinternal
subroutine merger_trees_render_dump
function gaussianregressionconstructorinternal
function posterioraspriorconstructorinternal
subroutine merger_tree_data_structure_-export_galacticus
subroutine merger_tree_data_structure_-export_irate
subroutine merger_tree_data_structure_-read_ascii
subroutine merger_tree_data_structure_-read_particles_ascii
subroutine differentialevolutionsimulate
subroutine particleswarmsimulate
subroutine stellar_population_-luminosity_tabulate
subroutine fspsreadfile
subroutine filereadfile
function standardinterpolate
file structure_formation.excursion_-sets.first_crossing_-distribution.Farahi.F90
function farahiconstructorinternal
subroutine farahifileread
subroutine takahashi2011lensingdistributionconstruct
function rodriguezpuebla2016constructorinternal
function tinker2008constructorinternal
file structure_formation.power_-spectrum.nonlinear.CosmicEmu.F90
subroutine restore_table
subroutine store_table
file structure_formation.transfer_-function.CAMB.F90
subroutine filereadfile
program test_diemerkravtsov2014_-concentration
program test_zhao2009_flat
program test_zhao2009_dark_energy
program test_zhao2009_open
program tests_halo_mass_function_-tinker
program test_stellar_populations_-luminosities
function io_hdf5_is_hdf5
subroutine io_hdf5_open_file
function xml_parse
function inputparametersconstructorfilechar
function inputparametersconstructornode
subroutine inputparametersfinalize
subroutine inputparametersparametersgroupopen

interface: `count_lines_in_file`

Description: Generic interface for `Count_Lines_in_File` function.
Code lines: 4
Contained by: module file_utilities

function: `count_lines_in_file_char`

Description: Returns the number of lines in the file `in_file` (version for character argument).
Code lines: 26
Contained by: module file_utilities
Modules used: galacticus_error

function: `count_lines_in_file_varstr`
Description: Returns the number of lines in the file `in_file` (version for varying string argument).
Code lines: 12
Contained by: module `file_utilities`

interface: `directory_make`
Description: Generic interface for functions that create a directory.
Code lines: 4
Contained by: module `file_utilities`

subroutine: `directory_make_char`
Description: Make the given directory path. Will create intermediate directories in the path if necessary.
Code lines: 20
Contained by: module `file_utilities`
Modules used: `galacticus_error`

subroutine: `directory_make_varstr`
Description: Make the given directory path. Will create intermediate directories in the path if necessary.
Code lines: 7
Contained by: module `file_utilities`

function: `executable_find`
Description: Return the full path to the executable of the given name.
Code lines: 38
Contained by: module `file_utilities`
Modules used: `iso_varying_string` `string_handling`

subroutine: `get_paths`
Description: Retrieve the `PATH` environment variable.
Code lines: 11
Contained by: function `executable_find`

interface: `file_exists`
Description: Generic interface for functions that check for a files existance.
Code lines: 4
Contained by: module `file_utilities`

function: `file_exists_char`
Description: Checks for existance of file `FileName` (version for character argument).
Code lines: 7
Contained by: module `file_utilities`

function: `file_exists_varstr`
Description: Checks for existance of file `FileName` (version for varying string argument).
Code lines: 7
Contained by: module `file_utilities`

subroutine: `file_lock`

Description: Place a lock on a file.
Code lines: 16
Contained by: module file_utilities

subroutine: file_lock_initialize
Description: Initilialize the per thread lock in a file lock object.
Code lines: 7
Contained by: module file_utilities

function: file_name
Description: Returns the path to the file.
Code lines: 12
Contained by: module file_utilities

function: file_name_expand
Description: Expands placeholders for Galacticus paths in file names.
Code lines: 12
Contained by: module file_utilities
Modules used: galacticus_paths

function: file_name_temporary
Description: Returns the path to the file.
Code lines: 27
Contained by: module file_utilities
Modules used: mpi_utilities string_handling

interface: file_path
Description: Generic interface for functions that return the path to a file.
Code lines: 4
Contained by: module file_utilities

function: file_path_char
Description: Returns the path to the file.
Code lines: 12
Contained by: module file_utilities

function: file_path_varstr
Description: Returns the path to the file.
Code lines: 8
Contained by: module file_utilities

subroutine: file_remove
Description: Remove a file.
Code lines: 8
Contained by: module file_utilities
Modules used: system_command

subroutine: file_unlock
Description: Remove a lock from a file.

Code lines: 17
Contained by: module file_utilities
Modules used: galacticus_error

type: lockdescriptor
Description: Type used to store file lock descriptors.
Code lines: 6
Contained by: module file_utilities

file: utility.hashes.F90
Description: Contains a module which implements "hashes" (i.e. associative arrays).
Code lines: 432

module: hashes
Description: Implements "hashes" (i.e. associative arrays). Derived type for TypeÂęlabel hashes. Constructor for scalar hashes. Destroys thisHash.
Code lines: 410
Contained by: file utility.hashes.F90
Modules used: iso_c_binding, iso_varying_string
Used by: module galacticus_meta_evolver_-profiler, module galacticus_nodes, subroutine treenodecomponentbuilder, module stellar_populations, program test_hashes, subroutine mpiinitialize

subroutine: delete_
Description: Deletes entry key from Hash.
Code lines: 23
Contained by: module hashes
Modules used: arrays_search, galacticus_error, iso_c_binding

subroutine: destroy_
Description: Destroys thisHash.
Code lines: 14
Contained by: module hashes

function: exists_
Description: Returns true if the specified key exists in the specified thisHash, false otherwise.
Code lines: 12
Contained by: module hashes

subroutine: initialize_
Description: Routine to initialize (or re-initialize) a hash.
Code lines: 10
Contained by: module hashes

function: key_
Description: Returns the key of entry number index in thisHash.
Code lines: 9

Contained by: module hashes

subroutine: keys_
Description: Returns an array of all keys in thisHash.
Code lines: 10
Contained by: module hashes

subroutine: set_
Description: Sets the value of key in thisHash to value.
Code lines: 65
Contained by: module hashes
Modules used: arrays_search iso_c_binding

function: size_
Description: Returns the number of elements in the specified Hash.
Code lines: 7
Contained by: module hashes

function: value_
Description: Returns the value of key in thisHash.
Code lines: 19
Contained by: module hashes
Modules used: arrays_search galacticus_error iso_c_binding

subroutine: values_
Description: Returns an array of all values in thisHash.
Code lines: 19
Contained by: module hashes
Modules used: galacticus_error

file: utility.hashes.cryptographic.F90
Code lines: 66

module: hashes_cryptographic
Code lines: 43
Contained by: file utility.hashes.cryptographic.F90
Modules used: iso_c_binding
Used by: subroutine interface_camb_transfer_-function program test_hashes_cryptographic
module input_parameters function inputparametersserializetostring

function: hash_md5
Code lines: 24
Contained by: module hashes_cryptographic
Modules used: iso_varying_string string_handling

file: utility.hashes.perfect.F90

Description: Contains a module which implements a perfect hash algorithm for long integer keys.
Code lines: 309

module: `hashes_perfect`
Description: Implements a perfact hash algorithm for long integer keys based on methods described by Czech et al. [1997]. The specific implementation follows the general structure of that given in a Dr. Dobbs article.
Code lines: 287
Contained by: file `utility.hashes.perfect.F90`
Modules used: `iso_c_binding` `kind_numbers`
Used by: program `test_perfect_hashes`

subroutine: `hash_perfect_create`
Description: Create a perfect hash for a given set of keys.
Code lines: 139
Contained by: module `hashes_perfect`
Modules used: `galacticus_error` `memory_management`

subroutine: `hash_perfect_destroy`
Description: Destroy a perfect hash.
Code lines: 11
Contained by: module `hashes_perfect`
Modules used: `memory_management`

function: `hash_perfect_index`
Description: Return the index corresponding to a hash key.
Code lines: 14
Contained by: module `hashes_perfect`
Modules used: `galacticus_error`

function: `hash_perfect_is_present`
Description: Returns true if the hash contains the key.
Code lines: 12
Contained by: module `hashes_perfect`
Modules used: `galacticus_error`

function: `hash_perfect_size`
Description: Return the size of the hash table.
Code lines: 10
Contained by: module `hashes_perfect`
Modules used: `galacticus_error`

function: `hash_perfect_value`
Description: Returns the value for a specified key.
Code lines: 13
Contained by: module `hashes_perfect`
Modules used: `galacticus_error`

type: `hashperfect`

Description: A derived type which stores perfect long integer hashes.
Code lines: 54
Contained by: module hashes_perfect

type: rowstructure
Description: A row structure used in building hashes
Code lines: 4
Contained by: module hashes_perfect

file: utility.input_parameters.F90
Description: Contains a module which implements reading of parameters from an XML data file.
Code lines: 1432

module: input_parameters
Description: Implements reading of parameters from an XML file.
Code lines: 1410
Contained by: file utility.input_parameters.F90
Modules used: fox_dom, function_classes, galacticus_build, galacticus_versioning, hashes_cryptographic, io_hdf5, io_xml, iso_varying_string, kind_numbers, string_handling
Used by: program galacticus; function bertschingerconstructorparameters; function coldmodeconstructorparameters; function naozbarkana2007constructorparameters; function simpleconstructorparameters; function bertschingerconstructorparameters; function simpleconstructorparameters; function zeroconstructorparameters; function adafconstructorparameters; function eddingtonlimitedconstructorparameters; function shakurasunyaevconstructorparameters; function hopkins2007constructorparameters; function switchedconstructorparameters; function vernerconstructorparameters; function vernerconstructorparameters; function sutherland1998constructorparameters; function vanhoof2014constructorparameters; function scholzwalters1991constructorparameters; function verner1996constructorparameters; function arnaud1985constructorparameters; function verner1996constructorparameters; program benchmark_stellar_populations_luminosities; function volonteri2003constructorinternal; function volonteri2003constructorparameters; function spheroidradiusfractionconstructorinternal; function spheroidradiusfractionconstructorparameters

function tidalradiusconstructorparameters
function zeroconstructorparameters
function zeroconstructorparameters
function hydrogennetworkconstructorparameters
function atomiccieclоudyconstructorparameters
function ciefileconstructorparameters
function cmbcomptonconstructorparameters
function molecularhydrogengallipallaconstructorparameters
subroutine summationdescriptor
function isothermalconstructorparameters
function cole2000constructorparameters
function cutoffconstructorparameters
function nocoolingsatellitesconstructorparameters
function simplescalingconstructorparameters
function zeroconstructorparameters
function darkmatterhaloconstructorparameters
function coolingfreefallconstructorparameters
function constantrotationconstructorparameters
function whitefrenk1991constructorparameters
function matterdarkenergyconstructorparameters
function staticuniverseconstructorparameters
function wdmthermalconstructorinternal
function correa2015constructorparameters
function zhao2009constructorparameters
function campanelli2007constructorparameters
function standardconstructorparameters
function rezzolla2008constructorparameters
function zeroconstructorparameters
function dynamicaltimeconstructorparameters
function cmbcomptonconstructorinternal
function atomiccieclоudyconstructorparameters
function summationconstructorparameters
function betaprofileconstructorparameters
function simpleconstructorparameters
function whitefrenk1991constructorparameters
function multiplierconstructorparameters
function simpleconstructorparameters
function velocitymaximumscalingconstructorparameters
function simpleconstructorparameters
function haloformationconstructorparameters
function coolingradiusconstructorparameters
function meanconstructorparameters
function formationtimeconstructorparameters
function matterlambdaconstructorparameters
function cdmconstructorparameters
function wdmthermalconstructorparameters
function wechsler2002constructorparameters
function vandenboschconstructorparameters

function zeroconstructorparameters
function bett2007constructorparameters
function massspinintegral
function lognormalconstructorparameters
function darkmatteronlyconstructorparameters
function summationconstructorparameters
function twobodyrelaxationconstructorparameters
function gao2008constructorparameters
function munozcuartas2011constructorparameters
function prada2011constructorparameters
function wdmconstructorparameters
function ludlow2014constructorparameters
function binaryconstructorparameters
function zeroconstructorparameters
function klypin2015constructorparameters
function einastoconstructorparameters
function heatedconstructorparameters
function isothermalconstructorparameters
function truncatedexponentialconstructorparameters
subroutine tasks_evolve_forest_-construct_null
function ismmassconstructorparameters
function alwaysconstructorparameters
function basicmassconstructorparameters
function haloisolatedconstructorparameters
function halonotisolatedconstructorparameters
function lightconeconstructorparameters
function virialdensitycontrastdefinitionparameters
function nbodyerrorsconstructorparameters
function deltafunctionconstructorparameters
function adiabaticgnedin2004constructorparameters
function nullconstructorparameters
function tidalconstructorparameters
function bullock2001constructorparameters
function ludlow2016fitconstructorparameters
function nfw1996constructorparameters
function schneider2015constructorparameters
function zhao2009constructorparameters
function ludlow2016constructorparameters
function concentrationconstructorparameters
function gao2008constructorparameters
function burkertconstructorparameters
function nfwconstructorparameters
function isothermalconstructorinternal
function truncatedconstructorparameters
subroutine node_promotion_index_shift
function virial_density_contrast_-percolation_objects_constructor_null
function allconstructorparameters
function anyconstructorparameters
function formationtimeconstructorparameters
function halomassconstructorparameters
function hostmassrangeconstructorparameters
function mainbranchconstructorparameters

function nodemajormergerrecentconstructorparameters
function notconstructorparameters
function rootnodeconstructorparameters
function spheroidstellarmassconstructorparameters
function starformationrateconstructorparameters
function stellarabsolutemagnitudesconstructorparameters
function stellarapparentmagnitudesconstructorparameters
function stellarmassconstructorparameters
function stellarmassmorphologyconstructorparameters
function surveygeometryconstructorparameters
function treehostedconstructorparameters
function efstathiou1982constructorparameters
function efstathiou1982tidalconstructorparameters
function fixedtimescaleconstructorparameters
function stableconstructorparameters
function equilibriumconstructorparameters
function fixedconstructorparameters
function linearconstructorparameters
function simpleconstructorparameters
subroutine galacticus_verbosity_set_from_parameters
subroutine galacticus_error_wait_set_from_parameters
subroutine galacticus_meta_evolver_profile
subroutine meta_tree_timing_initialize
subroutine galacticus_output_open_file
function hivshalomassrelationpadmanabhan2017constructorparameters
function localgroupmassfunctionconstructorparameters
function blackholebulgerelationconstructorparameters
function colordistributionsdssconstructorparameters
function concentrationdistributioncdmcococonstructorparameters
function conversionvshalomasscdmludlow2016constructorparameters
function correlationfunctionconstructorparameters
function correlationfunctionhearin2013sdssconstructorparameters
function binwidthconstructorparameters
function identityconstructorparameters
function log10tologconstructorparameters
function sequenceconstructorparameters
function unitarityconstructorparameters
function disksizeinclinationconstructorparameters
function grvtnllnsngconstructorparameters
function identityconstructorparameters
function randomerrorhialfalfaconstructorparameters
function randomerrornbdycncconstructorparameters
function randomerrornbodymassconstructorparameters
function randomerrorfixedconstructorparameters
function randomerrorpolynomialconstructorparameters
function sequenceconstructorparameters
function spinnbodyerrorsconstructorparameters
function galaxysizessdssconstructorparameters

function luminosityfunctionconstructorparameters
function luminosityfunctiongunawardhana2013sdssconstructorinternal
function luminosityfunctionsobral2013hizelsconstructorinternal
function luminosityfunctionmonterodorta2009sdssconstructorinternal
function massfunctionhialfalfamartin2010constructorinternal
function massfunctionhiconstructorparameters
function massfunctionstellarbernardi2013sdssconstructorinternal
function massfunctionstellarbaldry2012gamaconstructorinternal
function massfunctionstellarprimusconstructorinternal
function massfunctionstellarsdssconstructorinternal
function massfunctionstellarukidssudsconstructorinternal
function massfunctionstellarultravistaconstructorinternal
function massfunctionstellarvipersconstructorinternal
function massfunctionstellarzfourgeconstructorinternal
function massmetallicityandrews2013constructorparameters
function meanfunction1dconstructorparameters
function obreschkow2009constructorparameters
function multiconstructorparameters
function himassconstructorinternal
subroutine himassdestructor
function booleanconstructorparameters
function csmlgyluminositydistanceconstructorparameters
function filterhighpassconstructorparameters
function log10constructorparameters
function metallicity12lognhconstructorparameters
function luminosityfunctionhalphaconstructorparameters
function luminosityfunctiongunawardhana2013sdssconstructorparameters
function luminosityfunctionsobral2013hizelsconstructorparameters
function luminosityfunctionmonterodorta2009sdssconstructorparameters
function massfunctionhialfalfamartin2010constructorparameters
function massfunctionstellarbernardi2013sdssconstructorinternal
function massfunctionstellarconstructorparameters
function massfunctionstellarbaldry2012gamaconstructorparameters
function massfunctionstellarprimusconstructorparameters
function massfunctionstellarsdssconstructorparameters
function massfunctionstellarukidssudsconstructorparameters
function massfunctionstellarultravistaconstructorparameters
function massfunctionstellarvipersconstructorparameters
function massfunctionstellarzfourgeconstructorparameters
function massmetallicityblanc2017constructorparameters
function obreschkow2009constructorinternal
function morphologicalfractiongamamoffett2016constructorparameters
function nullconstructorparameters
function himassconstructorparameters
function antilog10constructorparameters
function csmlgyangulardistanceconstructorparameters
function filterhighpassconstructorinternal
function identityconstructorparameters
function magnitudeconstructorparameters
function minmaxconstructorinternal

function minmaxconstructorparameters
function multiplyconstructorparameters
function normalconstructorparameters
function squareconstructorparameters
function systmtcpolynomialconstructorparameters
function spindistributionbett2007constructorparameters
function volumefunction1dconstructorparameters
function csmlgyvolumeconstructorparameters
function filterhighpassconstructorparameters
function normalconstructorinternal
function propertyconstructorparameters
subroutine galacticus_output_halo_model_initialize
function squareconstructorparameters
function bernardi2013sdssconstructorinternal
function caputi2011ukidssudsconstructorparameters
function davidzon2013vipersconstructorparameters
function hearin2014sdssconstructorparameters
function liwhite2009sdssconstructorparameters
function localgroupsdssconstructorparameters
function martin2010alfalfaconstructorinternal
function monterodorta2009sdssconstructorparameters
function moustakas2013primusconstructorparameters
function tomczak2014zfourgeconstructorparameters
function fullskyconstructorparameters
function identityconstructorparameters
function multiplyconstructorinternal
function normalconstructorinternal
function sequenceconstructorparameters
function squarerootconstructorparameters
function scatterfunction1dconstructorparameters
function stellarvshalomassrelationleauthaud2012constructorparameters
function nbodymassconstructorparameters
function filterhighpassconstructorinternal
function identityconstructorparameters
function normalconstructorparameters
function sequenceconstructorparameters
subroutine state_initialize
function baldry2012gamaconstructorparameters
function bernardi2013sdssconstructorparameters
function davidzon2013vipersconstructorinternal
function gunawardhana2013sdssconstructorparameters
function kelvin2014gamanearconstructorparameters
function localgroupdesconstructorparameters
function localgroupclassicalconstructorparameters
function martin2010alfalfaconstructorparameters
function moustakas2013primusconstructorinternal
function muzzin2013ultravistaconstructorparameters
function combinedconstructorparameters
function behroozi2010constructorparameters
function triaxialityconstructorparameters

function virialradiusfractionconstructorinternal
function virialradiusfractionconstructorparameters
function enzohydrostaticconstructorparameters
function patejloeb2015constructorparameters
function ricotti2000constructorparameters
function betaprofileconstructorparameters
function growingconstructorparameters
function virialfractionconstructorinternal
function virialfractionconstructorparameters
function nullconstructorparameters
function henriques2013constructorparameters
function halodynamicaltimeconstructorparameters
function velocitymaximumscalingconstructorparameters
function zeroconstructorparameters
function font2008constructorinternal
function font2008constructorparameters
function zeroconstructorparameters
function font2008constructorinternal
function font2008constructorparameters
function halodynamicaltimeconstructorinternal
function halodynamicaltimeconstructorparameters
function rampressureaccelerationconstructorinternal
function rampressureaccelerationconstructorparameters
function virialradiusconstructorparameters
function enzohydrostaticconstructorparameters
function virialconstructorparameters
subroutine interface_camb_transfer_function
function gnedin2000constructorparameters
function recfastconstructorparameters
function fileconstructorparameters
function instantreionizationigmconstructorparameters
function internalconstructorparameters
function simpleigmconstructorparameters
function miyamotonagaiconstructorparameters
function exponentialdiskconstructorparameters
function hernquistconstructorparameters
function nfwconstructorparameters
function sersicconstructorparameters
function betaprofileconstructorparameters
function identityconstructorparameters
function inverseconstructorparameters
function logarithmconstructorparameters
function parkinson2008constructorinternal
function parkinson2008constructorparameters
function identityconstructorparameters
function buildconstructorparameters
function gaussianconstructorparameters
function halomassfunctionconstructorparameters
function powerlawconstructorparameters
function stellarmassfunctionconstructorparameters

function fixedmassconstructorparameters
function readhdf5constructorparameters
function readxmlconstructorparameters
function sampleddistributionconstructorparameters
function sampleddistributionpseudorandomconstructorparameters
function sampleddistributionquasirandomconstructorparameters
function sampleddistributionuniformconstructorparameters
function unionconstructorparameters
function fullyspecifiedconstructorparameters
subroutine readassignscaleradii
function readconstructorparameters
function sussingasciiconstructorparameters
function sussingconstructorparameters
function sussinghdf5constructorinternal
function sussinghdf5constructorparameters
function galacticusconstructorparameters
function smoothaccretionconstructorparameters
function staterestoredconstructorparameters
subroutine merger_tree_dump_evolution
subroutine merger_tree_structure_dump
function historyconstructorparameters
function multiconstructorparameters
function recordevolutionconstructorparameters
function satelliteconstructorparameters
function simpleconstructorparameters
function standardconstructorparameters
function nonevolvingconstructorparameters
function standardconstructorparameters
subroutine standardevolve
function standardtimeevolveto
function standardconstructorparameters
function singlelevelhierarchyconstructorparameters
function assignorbitsconstructorinternal
function assignorbitsconstructorparameters
function augmentconstructorparameters
subroutine conditionalmfoperate
function deforestconstructorparameters
function dumptographvizconstructorparameters
function exportconstructorparameters
subroutine exportoperate
subroutine massaccretionhistoryoperate
function monotonizemassgrowthconstructorparameters
function nullconstructorparameters
file merger_-trees.operators.particulate.F90
function particulateconstructorparameters
function perturbmassesconstructorinternal
function perturbmassesconstructorparameters
function profilerconstructorparameters
function prunebaryonsconstructorparameters
function prunebymassconstructorparameters
function pruneclonesconstructorparameters
function prunelightconeconstructorparameters

function selectwithinrangeconstructorparameters
subroutine merger_tree_structure_output
function multiconstructorparameters
function standardconstructorparameters
function allandformationnodesparameters
function allnodesbranchparameters
function isolatednodesbranchparameters
function fileconstructorparameters
file models.likelihoods.galaxy_-population.F90
function gaussianregressionconstructorparameters
function independentlikelihoodsconstructorparameters
function massfunctionconstructorparameters
function multivariatenormalstochasticconstructorparameters
function projectedcorrelationfunctionconstructorparameters
function activeconstructorparameters
function inactiveconstructorparameters
function gadgethdf5constructorparameters
function meanpositionconstructorinternal
function nullconstructorparameters
function rotationcurveconstructorparameters
function selfboundconstructorparameters
function sequenceconstructorparameters
function icmszconstructorparameters
function rateinfallcoldmodeconstructorparameters
function concentrationconstructorparameters
function ratecoolingconstructorparameters
function densitycontrastsconstructorparameters
function sequenceconstructorparameters
function analyzerconstructorparameters
function nullconstructorparameters
subroutine standardoutput
function allnodesparameters
function isolatednodesparameters
function treeconstructionparameters
function sedfitconstructorparameters
function galaxypopulationconstructorparameters
function halomassfunctionconstructorparameters
function independentlikelihoodssequentialconstructorparameters
function multivariatenormalconstructorparameters
function parameterspriorconstructorparameters
function spindistributionconstructorparameters
function derivedconstructorparameters
function gadgetbinaryconstructorparameters
function environmentaloverdensityconstructorparameters
function meanpositionconstructorparameters
function paircountsconstructorparameters
function selfboundconstructorinternal
function sequenceconstructorinternal
function velocitydispersionconstructorparameters
function icmxrayluminosityconstructorparameters
function concentrationconstructorinternal
function radiuscoolingconstructorparameters
function densityprofileconstructorparameters
function descendentsconstructorparameters

function finaldescendentconstructorparameters
function radiihalflightpropertiesconstructorparameters
function radiushalfmassconstructorparameters
function halobiasconstructorparameters
function haloenvironmentconstructorparameters
function masshostconstructorparameters
function indiceshostconstructorparameters
function fractionaccretionhotmodeconstructorparameters
function lightconeconstructorparameters
function lmnstyemssnlineconstructorparameters
function luminositystellarconstructorparameters
function lmnstystllrchrltfll2000constructorparameters
function mainbranchstatusconstructorparameters
function massismconstructorparameters
function massblackholeconstructorparameters
function masshaloconstructorinternal
function masshaloconstructorparameters
function massprofileconstructorparameters
function massstellarconstructorparameters
function massstellarmorphologyconstructorparameters
function massstellarspheroidconstructorparameters
function metallicityismconstructorparameters
function mostmassiveprogenitorconstructorparameters
function multiconstructorparameters
function nodeindicesconstructorparameters
function nullconstructorparameters
function projecteddensityconstructorparameters
function halfmassradiusconstructorparameters
function ratioconstructorparameters
function redshiftlastisolatedconstructorparameters
function rotationcurveconstructorparameters
function satelliteorbitalextremaconstructorparameters
function satellitestatusconstructorparameters
function spinbullockconstructorparameters
function spinconstructorparameters
function indicestreeconstructorparameters
function treeweightconstructorparameters
function velocitydispersionconstructorparameters
function velocitymaximumconstructorparameters
function virialpropertiesconstructorparameters
subroutine pseudorandominitialize
subroutine abundances_initialize
subroutine chemical_abundances_-initialize
function node_component_black_hole_-simple_seed_mass
function node_component_black_hole_-standard_seed_mass
function node_component_black_hole_-standard_seed_spin

subroutine nodeclasshierarchyinitialize
subroutine node_components_thread_initialize
subroutine node_component_basic_extended_bindings
subroutine node_component_black_hole_noncentral_initialize
subroutine node_component_black_hole_simple_initialize
subroutine node_component_black_hole_standard_initialize
subroutine node_component_dark_matter_profile_scale_initialize
subroutine node_component_dark_matter_profile_scale_shape_initialize
subroutine node_component_disk_standard_initialize
subroutine node_component_disk_very_simple_initialize
subroutine node_component_disk_very_simple_size_initialize
subroutine node_component_dynamics_statistics_bars_initialize
subroutine node_component_formation_times_cole2000_initialize
subroutine node_component_hot_halo_cold_mode_initialize
subroutine node_component_hot_halo_outflow_tracking_thread_initialize
subroutine node_component_hot_halo_standard_thread_initialize
subroutine node_component_hot_halo_vs_delayed_initialize
subroutine node_component_mass_flow_statistics_standard_initialize
subroutine node_component_merging_statistics_recent_initialize
subroutine node_component_merging_statistics_standard_initialize
subroutine node_component_nbody_generic_initialize
subroutine node_component_position_preset_orphans_initialize
subroutine node_component_position_trace_dark_matter_thread_initialize
subroutine node_components_initialize
subroutine node_component_age_statistics_standard_initialize
subroutine node_component_basic_extended_thread_initialize
subroutine node_component_black_hole_noncentral_thread_initialize
subroutine node_component_black_hole_simple_thread_initialize
subroutine node_component_black_hole_standard_thread_initialize
subroutine node_component_dark_matter_profile_scale_thread_initialize
subroutine node_component_dark_matter_profile_scale_shape_thread_init
subroutine node_component_disk_standard_thread_initialize
subroutine node_component_disk_very_simple_thread_initialize
subroutine node_component_disk_very_simple_size_thread_initialize
subroutine node_component_dynamics_statistics_bars_thread_initialize
subroutine node_component_formation_times_mass_fraction_initialize
subroutine node_component_hot_halo_cold_mode_thread_initialize
subroutine node_component_hot_halo_standard_initialize
subroutine node_component_hot_halo_very_simple_thread_initialize
subroutine node_component_interoutput_standard_thread_initialize
subroutine node_component_mass_flow_statistics_standard_thread_initialize
subroutine node_component_merging_statistics_recent_thread_initialize
subroutine node_component_merging_statistics_standard_thread_initialize
subroutine node_component_position_preset_initialize
subroutine node_component_position_preset_orphans_thread_initialize
subroutine node_component_satellite_orbiting_initialize

subroutine node_component_satellite_orbiting_thread_initialize
subroutine node_component_satellite_standard_thread_initialize
subroutine node_component_satellite_very_simple_thread_initialize
subroutine node_component_spheroid_standard_thread_initialize
subroutine node_component_spheroid_very_simple_thread_initialize
subroutine node_component_spin_vitvitska_thread_initialize
subroutine node_component_spin_random_thread_initialize
function listconstructorparameters
function likelihoodthresholdconstructorinternal
function neverconstructorparameters
function fixedconstructorparameters
function fixedconstructorparameters
function simpleconstructorparameters
function differentialevolutionconstructorparameters
function stochasticdifferentialevolutionconstructorparameters
function correlationconstructorparameters
function latinhypercubeconstructorparameters
function resumeconstructorparameters
function simpleconstructorparameters
function correlationlengthconstructorparameters
function stepcountconstructorinternal
function blackbodyconstructorparameters
function intergalacticbackgroundfileconstructorparameters
function nullconstructorparameters
function nullconstructorparameters
function nullconstructorparameters
function chandrasekhar1943constructorparameters
subroutine node_component_satellite_standard_initialize
subroutine node_component_satellite_very_simple_initialize
subroutine node_component_spheroid_standard_initialize
subroutine node_component_spheroid_very_simple_initialize
subroutine node_component_spin_vitvitska_bindings
subroutine node_component_spin_random_initialize
subroutine stellar_luminosities_initialize
function gelmanrubinconstructorparameters
function likelihoodthresholdconstructorparameters
function adaptiveconstructorparameters
function adaptiveconstructorparameters
function adaptiveconstructorparameters
function annealeddifferentialevolutionconstructorparameters
function particleswarmconstructorparameters
function reparameterizedifferentialevolutionconstructorparameters
function historyconstructorparameters
function priorrandomconstructorparameters
function switchedconstructorparameters
function correlationlengthconstructorinternal
function neverconstructorparameters
function stepcountconstructorparameters
function cosmicmicrowavebackgroundconstructorparameters
function intergalacticbackgroundinternalconstructorparameters
function summationconstructorparameters
function simpleconstructorparameters
function simpleconstructorparameters
function zeroconstructorparameters

function baugh2005constructorparameters
function verysimpleconstructorparameters
function cole2000constructorparameters
function standardconstructorparameters
function covington2008constructorparameters
function boylankolchin2008constructorparameters
function laceycole1993constructorparameters
function villalobos2013constructorparameters
function infiniteconstructorparameters
function randomconstructorparameters
function benson2005constructorparameters
function wetzel2010constructorparameters
function isotropicconstructorparameters
function tracedarkmatterconstructorparameters
function sphericalsymmetryconstructorparameters
function zeroconstructorparameters
function zeroconstructorparameters
function fixedconstructorparameters
function zeroconstructorparameters
function zeroconstructorparameters
function metallicitysplitconstructorparameters
function baugh2005constructorparameters
function fixedconstructorparameters
function intgrtdsurfacedensityconstructorparameters
function dynamicaltimeconstructorparameters
function sohalofinderparameters
function nbodyhalomasserrorfriendsoffriendsparameters
function simpleconstructorparameters
subroutine satellite_merging_output
function simpleconstructorparameters
function cole2000constructorparameters
function nullconstructorparameters
function jiang2008constructorparameters
function laceycole1993tormenconstructorparameters
function wetzelwhite2010constructorparameters
function presetconstructorparameters
function zeroconstructorparameters
function jiang2014constructorparameters
function fixedconstructorparameters
function spincorrelatedconstructorparameters
function nullconstructorparameters
function gnedin1999constructorparameters
function zentner2005constructorparameters
function fixedconstructorparameters
function superwindconstructorparameters
function superwindconstructorparameters
function insituconstructorparameters
function nullconstructorparameters
function dynamicaltimeconstructorparameters
function haloscalingconstructorparameters
function velocitymaxscalingconstructorparameters
function velocitymaxscalingconstructorparameters
function nbodyhalomasserrortrenti2010parameters
function nbodyhalomasserrornullparameters

function nbodyhalomasserrorpowerlawparameters
function cauchyconstructorparameters
function gammaconstructorparameters
function studenttconstructorparameters
function voightconstructorparameters
function betaconstructorparameters
function binomialconstructorparameters
function negativebinomialconstructorparameters
function lognormalconstructorparameters
function loguniformconstructorparameters
function negativeexponentialconstructorparameters
function normalconstructorparameters
function peakbackgroundconstructorparameters
function uniformconstructorparameters
function completeconstructorparameters
function surfacebrightnessconstructorparameters
function standardconstructorparameters
function fileconstructorparameters
function hegerwoosley2002constructorparameters
function nagashima2005constructorparameters
function fileconstructorparameters
function leitherer1992constructorparameters
function bpassconstructorparameters
function baugh2005topheavyconstructorparameters
function chabrier2001constructorparameters
function kennicutt1983constructorparameters
function kroupa2001constructorparameters
function millerscalo1979constructorparameters
function salpeter1955constructorparameters
function scalo1986constructorparameters
function piecewisepowerlawconstructorparameters
subroutine stellar_population_luminosity_tabulate
function instantaneousconstructorparameters
function noninstantaneousconstructorparameters
function diskspheroidconstructorparameters
function fixedconstructorparameters
function calzetti2000constructorparameters
function cardelli1989constructorparameters
function gordon2003constructorparameters
function prevotbouchetconstructorparameters
function wittgordon2003constructorparameters
function zeroconstructorparameters
subroutine tabulateddestructor
function inoue2014constructorparameters
function lycsuppressconstructorparameters
function madau1995constructorparameters
function meiksin2006constructorparameters
function lookupconstructorparameters

function identityconstructorparameters
function recentconstructorparameters
function sequenceconstructorparameters
subroutine sequencedescriptor
function unescapedconstructorparameters
function standardconstructorparameters
function standardinterpolate
function filteredpowerconstructorparameters
function variancepeakbackgroundsplitconstructorparameters
function kaiteyamasuto1996constructorparameters
function environmentalconstructorparameters
function fixedconstructorparameters
function peakbackgroundsplitconstructorparameters
function sphericalcollapsematterdeconstructorparameters
function sphericalcollapsematterlambdaconstructorparameters
function bardeen1986wdmconstructorparameters
function criticaloverdensityconstructorparameters
function linearconstructorparameters
function quadraticconstructorparameters
function remapshethmotormenconstructorparameters
function remapscaleconstructorparameters
function farahiconstructorinternal
function farahiconstructorparameters
function farahimidpointconstructorinternal
function farahimidpointconstructorparameters
function zhanghuiconstructorinternal
function zhanghuiconstructorparameters
function zhanghuihighorderconstructorinternal
function zhanghuihighorderconstructorparameters
function linearbarrierconstructorinternal
function linearbarrierconstructorparameters
function takahashi2011constructorparameters
function baryonicmodifierconstructorparameters
function pressschechterconstructorparameters
function sheth2001constructorparameters
function tinker2010constructorparameters
function lognormalconstructorparameters
function normalconstructorparameters
function uniformconstructorparameters
function bhattacharya2011constructorparameters
function despali2015constructorparameters
function pressschechterconstructorparameters
function rodriguezpuebla2016constructorparameters
function shethtormenconstructorparameters
function tinker2008constructorparameters
function tinker2008genericconstructorparameters
function environmentaveragedconstructorparameters
function environmentalconstructorparameters

function errorconvolvedconstructorparameters
function fofbiasconstructorparameters
function simplesystematicconstructorparameters
function simpleconstructorparameters
function cosmicemuconstructorparameters
function peacockdodds1996constructorparameters
function linearconstructorparameters
function simpleconstructorparameters
function standardconstructorparameters
function lagrangianchan2017constructorparameters
function sharpkspaceconstructorparameters
function smoothkspaceconstructorparameters
function tophatconstructorparameters
function tophatsharpkhybridconstructorparameters
subroutine make_table
function bbksconstructorparameters
function bbkswdmconstructorparameters
function bode2001constructorparameters
function cambconstructorinternal
function cambconstructorparameters
function eisensteinhu1999constructorparameters
function fileconstructorparameters
subroutine filereadfile
function identityconstructorinternal
function identityconstructorparameters
function giocoli2008constructorparameters
function bryannorman1998constructorparameters
function kitayamasuto1996constructorparameters
function fixedconstructorparameters
function friendsoffriendsconstructorinternal
function friendsoffriendsconstructorparameters
function percolationconstructorinternal
function percolationconstructorparameters
function virial_density_contrast_percolation_objects_constructor
function sphericalcollapsematterdeconstructorparameters
function sphericalcollapsematterlambdaconstructorparameters
subroutine system_limits_set
function localgroupdatabaseparameters
function agnspectrahopkins2008buildfileparameters
function buildtoolcambparameters
function buildtoolcloudyparameters
function buildtoolfspsparameters
function buildtoolrecfastparameters
function catalogprojectedcorrelationfunctionconstructorparameters
subroutine catalogprojectedcorrelationfunctionperform
function conditionalmassfunctionconstructorparameters
file tasks.evolve_forests.F90
subroutine tasks_evolve_forest_construct
function cyclicconstructorparameters
function fcfsconstructorparameters
function strideconstructorparameters
function excursionsetsconstructorparameters

function halomassfunctionconstructorparameters
function halomodelprojectedcorrelationfunctionconstructorparameters
function halomodelgenerateconstructorparameters
function halospindistributionconstructorparameters
function intergalacticmediumstateconstructorparameters
function massfunctioncovarianceconstructorparameters
function mergertreefilebuilderconstructorparameters
subroutine mergertreefilebuilderperform
function multiconstructorparameters
function nbodyanalyzeconstructorparameters
function posteriorsampleconstructorparameters
function powerspectraconstructorparameters
function reportparameters
program test_diemerkravtsov2014_concentration
program test_nfw96_concentration_dark_energy
program test_prada2011_concentration
program test_zhao2009_flat
program test_zhao2009_dark_energy
program test_zhao2009_open
program test_abundances
program tests_bug745815
program tests_comoving_distance
program test_correa2015_concentration
program test_cooling_functions
program tests_cosmic_age
program test_dark_matter_halo_radius_enclosing_mass
program test_dark_matter_profiles
program test_dark_matter_profiles_generic
program test_dark_matter_profiles_heated
program test_gaunt_factors
program tests_halo_mass_function_tinker
program test_hashes
program tests_kepler_orbits
program tests_linear_growth_eds
program tests_linear_growth_cosmological_constant
program tests_linear_growth_dark_energy
program tests_linear_growth_open
program test_correa2015_mah
program test_math_distributions
program test_nodes
program test_parameters
program tests_power_spectrum
program test_random
program tests_sigma
program tests_spherical_collapse_dark_energy_eds
program tests_spherical_collapse_dark_energy_omega_zero_point_six
program tests_spherical_collapse_dark_energy_omega_zero_point_eight
program tests_spherical_collapse_dark_energy_omega_half
program tests_spherical_collapse_dark_energy_omega_two_thirds
program tests_spherical_collapse_dark_energy_lambda
program tests_spherical_collapse_dark_energy_open
program tests_spherical_collapse_flat
program tests_spherical_collapse_nonlinear
program tests_spherical_collapse_open

program test_stellar_populations
program test_stellar_populations_luminosities
program tests_tree_branch_destroy
function nullconstructorparameters
function simpleconstructorparameters
function nullconstructorparameters
function simpleconstructorparameters
function identityconstructorparameters
function intergalacticmediumstateevolveconstructorparameters

type: `genericobjectlist`
Description: A list-type for unlimited polymorphic pointers.
Code lines: 4
Contained by: module input_parameters

type: `inputparameter`
Description: A class to handle input parameters for Galacticus.
Code lines: 69
Contained by: module input_parameters

subroutine: `inputparameterdestroy`
Description: Destructor for the `inputParameter` class.
Code lines: 17
Contained by: module input_parameters

function: `inputparameterget`
Description: Get the value of a parameter.
Code lines: 21
Contained by: module input_parameters
Modules used: galacticus_error

function: `inputparameterisparameter`
Description: Return true if this is a valid parameter.
Code lines: 13
Contained by: module input_parameters

interface: `inputparameterlist`
Description: Constructors for `inputParameterList` objects.
Code lines: 3
Contained by: module input_parameters

subroutine: `inputparameterlistadd`
Description: Add a parameter to a list of input parameters to an XML document.
Code lines: 25
Contained by: module input_parameters

function: `inputparameterlistconstructor`
Description: Construct an `inputParameterList` object.
Code lines: 7
Contained by: module input_parameters

subroutine: `inputparameterlistdestructor`
Description: Destroy an `inputParameterList` object.
Code lines: 8
Contained by: module `input_parameters`

subroutine: `inputparameterlistserializetoxml`
Description: Serialize a list of input parameters to an XML document.
Code lines: 15
Contained by: module `input_parameters`
Modules used: `fox_wxml`

function: `inputparameterobjectcreated`
Description: Return true if the specified instance of the object associated with this parameter has been created.
Code lines: 20
Contained by: module `input_parameters`

function: `inputparameterobjectget`
Description: Return a pointer to the object associated with this parameter.
Code lines: 22
Contained by: module `input_parameters`
Modules used: `galacticus_error`

subroutine: `inputparameterobjectset`
Description: Set a pointer to the object associated with this parameter.
Code lines: 23
Contained by: module `input_parameters`

subroutine: `inputparameterreset`
Description: Reset objects associated with this parameter and any sub-parameters.
Code lines: 26
Contained by: module `input_parameters`

interface: `inputparameters`
Description: Constructors for the `inputParameters` class.
Code lines: 7
Contained by: module `input_parameters`

subroutine: `inputparametersaddparameter`
Description: Add a parameter to the set.
Code lines: 36
Contained by: module `input_parameters`

subroutine: `inputparametersbuildtree`
Description: Build a tree representation of the input parameter file.
Code lines: 32
Contained by: module `input_parameters`

subroutine: `inputparameterscheckparameters`
Code lines: 95
Contained by: module `input_parameters`
Modules used: `galacticus_display` `regular_expressions` `string_handling`

function: `inputparametersconstructorcopy`
Description: Constructor for the `inputParameters` class from an existing parameters object.
Code lines: 11
Contained by: module `input_parameters`

function: `inputparametersconstructorfilechar`
Description: Constructor for the `inputParameters` class from an XML file specified as a character variable.
Code lines: 30
Contained by: module `input_parameters`
Modules used: `file_utilities` `galacticus_error` `io_xml`

function: `inputparametersconstructorfilevarstr`
Description: Constructor for the `inputParameters` class from an XML file specified as a variable length string.
Code lines: 12
Contained by: module `input_parameters`

function: `inputparametersconstructornode`
Description: Constructor for the `inputParameters` class from an FoX node.
Code lines: 81
Contained by: module `input_parameters`
Modules used: `file_utilities` `galacticus_display` `galacticus_error` `galacticus_paths`

function: `inputparametersconstructornull`
Description: Constructor for the `inputParameters` class creating a null instance.
Code lines: 12
Contained by: module `input_parameters`

function: `inputparametersconstructorvarstr`
Description: Constructor for the `inputParameters` class from an XML file specified as a variable length string.
Code lines: 22
Contained by: module `input_parameters`

function: `inputparameterscopiescount`
Description: Return true if the specified parameter is present.
Code lines: 32
Contained by: module `input_parameters`
Modules used: `galacticus_error`

function: `inputparameterscount`
Description: Return a count of the number of values in a parameter.
Code lines: 22
Contained by: module `input_parameters`
Modules used: `galacticus_error`

subroutine: `inputparametersdestroy`
Description: Destructor for the `inputParameters` class.
Code lines: 17
Contained by: module `input_parameters`

subroutine: `inputparametersetdouble`
Description: Set the value of a parameter.
Code lines: 10
Contained by: module `input_parameters`

subroutine: `inputparametersetvarstr`
Description: Set the value of a parameter.
Code lines: 8
Contained by: module `input_parameters`

subroutine: `inputparametersfinalize`
Description: Finalizer for the `inputParameters` class.
Code lines: 34
Contained by: module `input_parameters`
Modules used: `file_utilities`

function: `inputparametersisglobal`
Description: Return true if an `inputParameters` object is the global input parameter set.
Code lines: 7
Contained by: module `input_parameters`

function: `inputparametersispresent`
Description: Return true if the specified parameter is present.
Code lines: 27
Contained by: module `input_parameters`

subroutine: `inputparametersmarkglobal`
Description: Mark an `inputParameters` object as the global input parameters.
Code lines: 13
Contained by: module `input_parameters`
Modules used: `galacticus_error`

function: `inputparametersnode`
Description: Return the node containing the parameter.
Code lines: 35
Contained by: module `input_parameters`

Modules used: galacticus_error

subroutine: inputparametersparametersgroupcopy
Description: Copy an output group for parameters in the given HDF5 object.
Code lines: 11
Contained by: module input_parameters

subroutine: inputparametersparametersgroupopen
Description: Open an output group for parameters in the given HDF5 object.
Code lines: 25
Contained by: module input_parameters
Modules used: file_utilities

subroutine: inputparametersreset
Description: Reset all objects in a parameter set.
Code lines: 7
Contained by: module input_parameters

subroutine: inputparametersresolvereferences
Description: Build a tree representation of the input parameter file.
Code lines: 24
Contained by: module input_parameters

function: inputparametersserializetostring
Description: Serialize input parameters to a string.
Code lines: 32
Contained by: module input_parameters
Modules used: hashes_cryptographic

subroutine: inputparametersserializetoxml
Description: Serialize input parameters to an XML file.
Code lines: 8
Contained by: module input_parameters

function: inputparameterssubparameters
Description: Return sub-parameters of the specified parameter.
Code lines: 29
Contained by: module input_parameters
Modules used: galacticus_error

subroutine: inputparametersvalidatename
Description: Validate a parameter name.
Code lines: 10
Contained by: module input_parameters
Modules used: galacticus_error

subroutine: inputparametersvaluename
Description: Return the value of the parameter specified by name.
Code lines: 31

Contained by: module input_parameters
Modules used: galacticus_error

subroutine: inputparametersvaluenode
Description: Return the value of the specified parameter.
Code lines: 65
Contained by: module input_parameters
Modules used: galacticus_error

function: inputparameterswalktree
Description: Perform a depth-first walk of a parameter tree.
Code lines: 23
Contained by: module input_parameters

file: utility.kind_numbers.F90
Description: Contains a module which defines various kind types.
Code lines: 34

module: kind_numbers
Description: Defines various kind types.
Code lines: 13
Contained by: file utility.kind_numbers.F90
Used by:

file accretion.halo.cold_mode.F90	program benchmark_stellar_populations_luminosities
file cooling.cooling_radius.beta_profile.F90	file cooling.cooling_radius.isothermal_profile.F90
file cooling.cooling_radius.simple.F90	file cooling.specific_angular_momentum.constant_rotation.F90
file dark_matter_halos.scales.virial_density_contrast.F90	module dark_matter_profiles
file dark_matter_profiles_DMO.Burkert.F90	file dark_matter_profiles_DMO.Einasto.F90
file dark_matter_profiles_DMO.NFW.F90	file dark_matter_profiles_DMO.heated.F90
module node_events_inter_tree	function galactic_structure_radius_enclosing_mass
module galactic_structure_potentials	module galacticus_meta_tree_timing
subroutine galacticus_extra_output_halo_fourier_profile	file hot_halo.outflow_reincorporation.velocity_maximum_scaling.F90
file hot_halo.ram_pressure_stripping.Font2008.F90	function erfapproximatequad
function buildconstruct	subroutine cole2000build
function fullyspecifiedconstruct	function indexnode
function nodelookup	file merger_trees.construct.read.F90
module merger_tree_read_importers	subroutine sussingasciiload

subroutine sussingtreeindicesread
function smoothaccretionconstruct
subroutine mergertreestatestore
subroutine evolve_to_time_report
file merger_trees.evolver.standard.F90

file merger_-
trees.operators.information_-
content.F90
file merger_-
trees.operators.profiler.F90
subroutine regridtimesoperate

module merger_trees_render
file nodes.property_extractor.integer_-
scalar.F90
module numerical_constants_math
function interpolate_linear_generate_-
factors
module arrays_search
function search_indexed_integer8
subroutine sort_do_integer8
subroutine sort_do_integer8_c
subroutine sort_index_do_double_c
function sort_index_do_integer8
subroutine sort_index_do_integer_c
module nbody_simulation_data
subroutine kepler_orbits_output
module merger_trees_dump
function satellitepresetnodeindex

subroutine node_component_black_hole_-
standard_output
subroutine node_component_disk_-
standard_star_formation_history_-
output
module node_component_hot_halo_-
standard
subroutine node_component_mass_flow_-
statistics_standard_extra_output
subroutine node_component_merging_-
statistics_recent_output
subroutine node_component_nbody_-
generic_set_integer_property

subroutine sussinghdf5load
subroutine mergertreestatefromfile
function nodearrayposition
module merger_trees_evolve
file merger_trees.node_-
evolver.standard.F90
file merger_-
trees.operators.particulate.F90

file merger_trees.operators.prune_non_-
essential.F90
file merger_-
trees.outputter.standard.F90
function galaxypopulationevaluate
file nodes.property_extractor.integer_-
tuple.F90
function interpolate_linear_do
function interpolate_locate

function search_array_integer8
subroutine sort_do_double_both
subroutine sort_do_integer8_both
function sort_index_do_double
function sort_index_do_integer
subroutine sort_index_do_integer8_c
subroutine sortindex
module histories
module merger_tree_data_structure
module galacticus_nodes
subroutine node_component_black_hole_-
simple_output
subroutine node_component_black_hole_-
standard_output_properties
module node_component_disk_standard_-
data

subroutine node_component_inter_-
output_standard_reset
subroutine node_component_merging_-
statistics_major_output
subroutine node_component_nbody_-
generic_output
subroutine node_component_spheroid_-
standard_star_formation_history_-
output

subroutine stellar_luminosities_output
function listtimenext

subroutine differentialevolutionsimulate
subroutine particleswarmsimulate

module satellite_orbits

module star_formation_histories

file star_formation.rate_surface_density.disks.Kennicutt-Schmidt.F90
file star_formation.timescales.disks.velocity_maximum_scaling.F90
subroutine statistics_points_correlation
subroutine farahiratetabulate
subroutine farahimidpointratetabulate
file structure_formation.halo_environment.normal.F90
subroutine make_table

file tasks.evolve_forests.F90
program tests_io_hdf5
program tests_bug745815
program test_integration2
program test_search
program test_string_utilities
program test_vectors

subroutine io_hdf5_read_attribute_integer8_1d_array_static
subroutine io_hdf5_read_dataset_integer8_1d_array_allocatable
subroutine io_hdf5_read_dataset_integer8_2d_array_allocatable
subroutine io_hdf5_read_table_integer8_1d_array_allocatable
subroutine io_hdf5_write_attribute_integer8_scalar
subroutine io_hdf5_write_dataset_integer8_2d

function listindex
subroutine differentialevolutionposterior
subroutine particleswarmposterior

function intergalacticbackgroundinternalupdate
file satellites.tidal_stripping.rate.Zentner2005.F90
file star_formation.rate_surface_density.disks.Blitz2006.F90
file star_formation.rate_surface_density.disks.Krumholz2009.F90
file star_formation.timescales.spheroids.velocity_maximum_scaling.F90
function farahiprobability

function farahimidpointprobability
function lognormaloverdensitylinear
function normaloverdensitylinear

file structure_formation.virial_density_contrast.percolation.F90
subroutine evolveforestssuspendtree
program test_array_monotonicity
program test_perfect_hashes
program test_math_fast
program test_sort
program tests_tree_branch_destroy
subroutine io_hdf5_read_attribute_integer8_1d_array_allocatable
subroutine io_hdf5_read_attribute_integer8_scalar
subroutine io_hdf5_read_dataset_integer8_1d_array_static
subroutine io_hdf5_read_dataset_integer8_2d_array_static
subroutine io_hdf5_write_attribute_integer8_1d
subroutine io_hdf5_write_dataset_integer8_1d
function array_index_integer8

function array_is_monotonic_integer8
module hashes_perfect
module input_parameters
module memory_management
function concatenate_varstr_integer8
subroutine assert_integer8_1d_array
subroutine assert_integer8_scalar

file: utility.locks.F90
Description: Contains a module which implements advanced locks.
Code lines: 375

module: locks
Description: Provides advanced locks.
Code lines: 353
Contained by: file utility.locks.F90
Modules used: iso_c_binding
Used by: file galacticus.output.analyses.distribution_operator.gravitational_lensing.F90
file structure_formation.gravitational_lensing.Takahashi_2011.F90
module io_hdf5
module stellar_population_luminosities
program test_locks

interface: ompincrementallock
Description: Interface to constructors for OpenMP incremental locks.
Code lines: 3
Contained by: module locks

function: ompincrementallockconstructor
Description: Constructor for OpenMP incremental lock objects.
Code lines: 7
Contained by: module locks

subroutine: ompincrementallockdestructor
Description: Destructor for OpenMP incremental lock objects.
Code lines: 8
Contained by: module locks

subroutine: ompincrementallockinitialize
Description: (Re)initialize an OpenMP incremental lock object.
Code lines: 9
Contained by: module locks

subroutine: ompincrementallockset
Description: Get a lock on an OpenMP incremental lock object.
Code lines: 14
Contained by: module locks

subroutine: ompincrementallockunset
Description: Release a lock on an OpenMP incremental lock object.

Code lines: 7
Contained by: module locks

interface: omplock
Description: Interface to constructors for OpenMP locks.
Code lines: 3
Contained by: module locks

function: omplockconstructor
Description: Constructor for OpenMP lock objects.
Code lines: 7
Contained by: module locks

subroutine: omplockdestructor
Description: Destructor for OpenMP lock objects.
Code lines: 8
Contained by: module locks

subroutine: omplockinitialize
Description: (Re)initialize an OpenMP lock object.
Code lines: 9
Contained by: module locks

function: omplockownedbythread
Description: Return true if the lock is owend by the current thread.
Code lines: 8
Contained by: module locks

subroutine: omplockset
Description: Get a lock on an OpenMP lock objects.
Code lines: 9
Contained by: module locks

subroutine: omplockunset
Description: Release a lock on an OpenMP lock objects.
Code lines: 8
Contained by: module locks

interface: ompreadwritelock
Description: Interface to constructors for OpenMP read/write locks.
Code lines: 3
Contained by: module locks

function: ompreadwritelockconstructor
Description: Constructor for OpenMP read/write lock objects.
Code lines: 9
Contained by: module locks

subroutine: ompreadwritelockdestructor

Description: Destructor for OpenMP read/write lock objects.
Code lines: 15
Contained by: module `locks`

subroutine: `ompreadwritelockinitialize`
Description: (Re)initialize an OpenMP read/write lock object.
Code lines: 11
Contained by: module `locks`

subroutine: `ompreadwritelocksetread`
Description: Get a read lock on an OpenMP read/write lock objects.
Code lines: 7
Contained by: module `locks`

subroutine: `ompreadwritelocksetwrite`
Description: Get a write lock on an OpenMP read/write lock objects.
Code lines: 16
Contained by: module `locks`

subroutine: `ompreadwritelockunsetread`
Description: Release a read lock on an OpenMP read/write lock objects.
Code lines: 7
Contained by: module `locks`

subroutine: `ompreadwritelockunsetwrite`
Description: Release a write lock on an OpenMP read/write lock objects.
Code lines: 14
Contained by: module `locks`

file: `utility.memory_management.F90`
Description: Contains a module for storing and reporting memory usage by the code.
Code lines: 2303

module: `memory_management`

Description:	Routines and data type for storing and reporting on memory usage. Also contains routines for allocating and deallocating arrays with automatic error checking and deallocation at program termination and memory usage reporting. Contains interface and type definitions for memory management functions along with storage space for pointers and sizes. This file was created automatically by `memoryUseageFunctions.pl` Contains memory management functions. This file was created automatically by `memoryUseageFunctions.pl`	
Code lines:	2281	
Contained by:	file utility.memory_management.F90	
Modules used:	galacticus_error	iso_c_binding
	kind_numbers	
Used by:	subroutine filedestructor	subroutine atomic_data_initialize
	subroutine matterdarkenergymakeexpansionfactortable	subroutine matterlambdamakedistancetable
	subroutine matterlambdamakeexpansionfactortable	subroutine nbodyerrorstabulate
	subroutine einastoenergytablemake	subroutine einastofourierprofiletablemake
	subroutine einastofreefalltabulate	subroutine einastoradialvelocitydispersiontabulate

subroutine einastoradiusfromspecificangularmomentumtabulate
function stellarabsolutemagnitudesconstructorparameters
function stellarapparentmagnitudesconstructorparameters
subroutine radiussolve
subroutine galacticus_meta_evolver_-profile
subroutine meta_tree_timing_post_tree_-evolve
function hivshalomassrelationpadmanabhan2017constructorinternal
function blackholemassrelationconstructorinternal
function colordistributionsdssconstructorinternal
function concentrationdistributioncdmcococonstructorinternal
function concentrationvshalomasscdmludlow2016constructorinternal
function correlationfunctionconstructorinternal
subroutine correlationfunctionfinalizeanalysis
function grvtnllnsngoperatedistribution
function galaxysizessdssconstructorinternal
function luminosityfunctionconstructorinternal
function luminosityfunctionhalphaconstructorinternal
function massfunctionhiconstructorinternal
function massfunctionstellarconstructorinternal
function massmetallicityandrews2013constructorinternal
function massmetallicityblanc2017constructorinternal
function meanfunction1dconstructorinternal
function meanfunction1dconstructorparameters
subroutine meanfunction1dresults
function morphologicalfractiongamamoffett2016constructorinternal
function csmlgyangulardistanceconstructorinternal
subroutine csmlgyangulardistancedestructor
function csmlgyluminositydistanceconstructorinternal
subroutine csmlgyluminositydistancedestructor
function scatterfunction1dconstructorinternal
function scatterfunction1dconstructorparameters
function spindistributionbett2007constructorinternal
function stellarvshalomassrelationleauthaud2012constructorinternal
function volumefunction1dconstructorinternal
function volumefunction1dconstructorparameters
subroutine volumefunction1dresults
subroutine galacticus_extra_output_-halo_fourier_profile
function squareconstructorinternal
function squareisinlightcone
subroutine caputi2011ukidssudsrandomsinitialize
subroutine liwhite2009sdssrandomsinitialize
subroutine martin2010alfalfarandomsinitialize
subroutine fullskywindowfunctions
function mangleangularpower
subroutine randompointswindowfunctions
subroutine halo_model_projected_-correlation

subroutine filter_response_load
subroutine sersictabulate
function buildconstructorparameters
function fixedmassconstructorparameters
subroutine sampleddistributionconstruct
function fullyspecifiedconstruct

function readconstruct
subroutine readcreatenodearray
subroutine readdestroynodeindices

subroutine sussingasciiload
subroutine sussingimport
subroutine sussinghdf5destructor
subroutine sussinghdf5open
function historyconstructorinternal

subroutine standardevolve
function augmentconstructorparameters
function conditionalmfconstructorinternal
subroutine conditionalmffinalize
subroutine exportoperate
subroutine massaccretionhistoryoperate
function profilerconstructorinternal
subroutine merger_tree_structure_output
subroutine standardextenddoublebuffer
subroutine standardoutputgroupcreate
function sedfitconstructorinternal
subroutine gaussianregressionfunctionchanged
function polynomialiteratorconstructor

function massfunctionconstructorinternal
subroutine posteriorasprioriniti alize

function spindistributionconstructorinternal
subroutine environmentaloverdensityoperate
subroutine paircountsoperate
subroutine selfboundoperate
function lmnstyemssnlineconstructorinternal

subroutine internalstateset
subroutine buildconstructmasses
subroutine fixedmassconstruct
subroutine readconstruct
subroutine unionconstruct
subroutine readbuildchildandsiblinglinks
function readconstructorinternal
subroutine readcreatenodeindices
subroutine readrootnodeaffinitiesinitial
subroutine sussingasciiopen
subroutine sussingtreeindicesread
subroutine sussinghdf5load
subroutine galacticusimport
function recordevolutionconstructorinternal
function augmentconstructorinternal
subroutine augmentfinalize
function conditionalmfconstructorparameters
subroutine conditionalmfoperate
subroutine informationcontentoperate
subroutine particulateoperate
function regridtimesconstructorinternal
subroutine standardbuffersallocate
subroutine standardextendintegerbuffer
subroutine merger_trees_render_dump
function gaussianregressionevaluate
subroutine gaussianregressionrestore

function halomassfunctionconstructorinternal
function posterioraspriorconstructorinternal
function projectedcorrelationfunctionconstructorinternal
function gadgetbinaryimport

subroutine meanpositionoperate

subroutine rotationcurveoperate
subroutine velocitydispersionoperate
function luminositystellarconstructorinternal

subroutine luminositystellardestructor
function lmnstystllrchrltfll2000constructorinternal
subroutine compositegausskronrod1dinitialize
subroutine integratormultidestructor
subroutine multivectorizedcompositegausskronrod1dinitialize
subroutine multivectorizedcompositetrapezoidal1dinitialize
subroutine tolerancessetgeneric
subroutine vectorizedcompositegausskronrod1dinitialize
subroutine vectorizedcompositetrapezoidalinitialize1d
function interpolate_2d_irregular_array
subroutine nearestneighborssearchfixedradius
subroutine points_prune
subroutine points_replicate
subroutine points_survey_geometry
subroutine abundances_allocate_elemental_values
subroutine abundances_destroy
subroutine abundances_initialize
subroutine chemical_abundances_allocate_values
subroutine chemical_abundances_initialize
subroutine chemicals_abundances_destroy
subroutine chemicals_read_raw
subroutine history_append_epoch
subroutine history_append_history
subroutine history_clone
subroutine history_create
subroutine history_destroy
subroutine history_long_integer_append_epoch
subroutine history_long_integer_append_history
subroutine history_long_integer_clone
subroutine history_long_integer_create
subroutine history_long_integer_destroy
subroutine history_long_integer_read_raw
subroutine history_long_integer_trim
subroutine history_long_integer_trim_forward
subroutine history_read_raw
subroutine history_timesteps
subroutine history_trim
subroutine history_trim_forward
subroutine merger_tree_data_construct_particle_indices
subroutine merger_tree_data_structure_add_metadata
subroutine merger_tree_data_structure_export_galacticus
subroutine merger_tree_data_structure_export_irate
subroutine merger_tree_data_structure_read_ascii
subroutine merger_tree_data_structure_read_particles_ascii
subroutine merger_tree_data_structure_reset
subroutine merger_tree_data_structure_set_particle_property_column
subroutine merger_tree_data_structure_set_property_column
subroutine merger_tree_data_structure_set_property_double
subroutine merger_tree_data_structure_set_property_integer8
subroutine merger_tree_data_structure_set_tree_indices
module galacticus_nodes
subroutine dynamicsstatisticsbarsadiabaticratioset

subroutine dynamicsstatisticsbarsbarinstabilitytimeset
subroutine dynamicsstatisticsbarstimeset
subroutine mergingstatisticsmajormajormergertimeset
subroutine mergingstatisticsrecentrecentmajormergercountset
subroutine nbodygenericintegersset
subroutine nbodygenericrealsset
subroutine node_component_dynamics_statistics_bars_record
subroutine nodeclasshierarchyinitialize
subroutine nodecomponentagestatisticsnullbuilder
subroutine nodecomponentagestatisticsnulldeserializeraw
subroutine nodecomponentagestatisticsnullfinalize
subroutine nodecomponentagestatisticsstandardbuilder
subroutine nodecomponentagestatisticsstandarddeserializeraw
subroutine nodecomponentagestatisticsstandardfinalize
subroutine nodecomponentbasicextendedtrackingbuilder
subroutine nodecomponentbasicextendedtrackingdeserializeraw
subroutine nodecomponentbasicextendedtrackingfinalize
subroutine nodecomponentbasicnonevolvingbuilder
subroutine nodecomponentbasicnonevolvingdeserializeraw
subroutine nodecomponentbasicnonevolvingfinalize
subroutine nodecomponentbasicnullbuilder
subroutine nodecomponentbasicnulldeserializeraw
subroutine nodecomponentbasicnullfinalize
subroutine nodecomponentbasicstandardbuilder
subroutine nodecomponentbasicstandarddeserializeraw
subroutine nodecomponentbasicstandardextendedbuilder
subroutine nodecomponentbasicstandardextendeddeserializeraw
subroutine nodecomponentbasicstandardextendedfinalize
subroutine nodecomponentbasicstandardfinalize
subroutine nodecomponentbasicstandardtrackingbuilder
subroutine nodecomponentbasicstandardtrackingdeserializeraw
subroutine nodecomponentbasicstandardtrackingfinalize
subroutine nodecomponentblackholenoncentralbuilder
subroutine nodecomponentblackholenoncentraldeserializeraw
subroutine nodecomponentblackholenoncentralfinalize
subroutine nodecomponentblackholenullbuilder
subroutine nodecomponentblackholenulldeserializeraw
subroutine nodecomponentblackholenullfinalize
subroutine nodecomponentblackholesimplebuilder
subroutine nodecomponentblackholesimpledeserializeraw
subroutine nodecomponentblackholesimplefinalize
subroutine nodecomponentblackholestandardbuilder
subroutine nodecomponentblackholestandarddeserializeraw
subroutine nodecomponentblackholestandardfinalize
subroutine nodecomponentdarkmatterprofilenullbuilder
subroutine nodecomponentdarkmatterprofilenulldeserializeraw

subroutine nodecomponentdarkmatterprofilenullfinalize
subroutine nodecomponentdarkmatterprofilescaledeserializeraw
subroutine nodecomponentdarkmatterprofilescalepresetbuilder
subroutine nodecomponentdarkmatterprofilescalepresetfinalize
subroutine nodecomponentdarkmatterprofilescaleshapedeserializeraw
subroutine nodecomponentdisknullbuilder
subroutine nodecomponentdisknullfinalize
subroutine nodecomponentdiskstandarddeserializeraw
subroutine nodecomponentdiskverysimplebuilder
subroutine nodecomponentdiskverysimplefinalize
subroutine nodecomponentdiskverysimplesizedeserializeraw
subroutine nodecomponentdynamicsstatisticsbarsbuilder
subroutine nodecomponentdynamicsstatisticsbarsfinalize
subroutine nodecomponentdynamicsstatisticsnulldeserializeraw
subroutine nodecomponentformationtimecole2000builder
subroutine nodecomponentformationtimecole2000finalize
subroutine nodecomponentformationtimemassfractiondeserializeraw
subroutine nodecomponentformationtimenullbuilder
subroutine nodecomponentformationtimenullfinalize
subroutine nodecomponenthosthistorynulldeserializeraw
subroutine nodecomponenthosthistorystandardbuilder
subroutine nodecomponenthosthistorystandardfinalize
subroutine nodecomponentdarkmatterprofilescalebuilder
subroutine nodecomponentdarkmatterprofilescalefinalize
subroutine nodecomponentdarkmatterprofilescalepresetdeserializeraw
subroutine nodecomponentdarkmatterprofilescaleshapebuilder
subroutine nodecomponentdarkmatterprofilescaleshapefinalize
subroutine nodecomponentdisknulldeserializeraw
subroutine nodecomponentdiskstandardbuilder
subroutine nodecomponentdiskstandardfinalize
subroutine nodecomponentdiskverysimpledeserializeraw
subroutine nodecomponentdiskverysimplesizebuilder
subroutine nodecomponentdiskverysimplesizefinalize
subroutine nodecomponentdynamicsstatisticsbarsdeserializeraw
subroutine nodecomponentdynamicsstatisticsnullbuilder
subroutine nodecomponentdynamicsstatisticsnullfinalize
subroutine nodecomponentformationtimecole2000deserializeraw
subroutine nodecomponentformationtimemassfractionbuilder
subroutine nodecomponentformationtimemassfractionfinalize
subroutine nodecomponentformationtimenulldeserializeraw
subroutine nodecomponenthosthistorynullbuilder
subroutine nodecomponenthosthistorynullfinalize
subroutine nodecomponenthosthistorystandarddeserializeraw
subroutine nodecomponenthothalocoldmodebuilder

subroutine nodecomponenthothalocoldmodedeserializeraw
subroutine nodecomponenthothalocoldmodefinalize
subroutine nodecomponenthothalonullbuilder
subroutine nodecomponenthothalonulldeserializeraw
subroutine nodecomponenthothalonullfinalize
subroutine nodecomponenthothalooutflowtrackingbuilder
subroutine nodecomponenthothalooutflowtrackingdeserializeraw
subroutine nodecomponenthothalooutflowtrackingfinalize
subroutine nodecomponenthothalostandardbuilder
subroutine nodecomponenthothalostandarddeserializeraw
subroutine nodecomponenthothalostandardfinalize
subroutine nodecomponenthothaloverysimplebuilder
subroutine nodecomponenthothaloverysimpledelayedbuilder
subroutine nodecomponenthothaloverysimpledelayeddeserializeraw
subroutine nodecomponenthothaloverysimpledelayedfinalize
subroutine nodecomponenthothaloverysimpledeserializeraw
subroutine nodecomponenthothaloverysimplefinalize
subroutine nodecomponentindicesnullbuilder
subroutine nodecomponentindicesnulldeserializeraw
subroutine nodecomponentindicesnullfinalize
subroutine nodecomponentindicesstandardbuilder
subroutine nodecomponentindicesstandarddeserializeraw
subroutine nodecomponentindicesstandardfinalize
subroutine nodecomponentinteroutputnullbuilder
subroutine nodecomponentinteroutputnulldeserializeraw
subroutine nodecomponentinteroutputnullfinalize
subroutine nodecomponentinteroutputstandardbuilder
subroutine nodecomponentinteroutputstandarddeserializeraw
subroutine nodecomponentinteroutputstandardfinalize
subroutine nodecomponentmassflowstatisticsnullbuilder
subroutine nodecomponentmassflowstatisticsnulldeserializeraw
subroutine nodecomponentmassflowstatisticsnullfinalize
subroutine nodecomponentmassflowstatisticsstandardbuilder
subroutine nodecomponentmassflowstatisticsstandarddeserializeraw
subroutine nodecomponentmassflowstatisticsstandardfinalize
subroutine nodecomponentmergingstatisticsmajorbuilder
subroutine nodecomponentmergingstatisticsmajordeserializeraw
subroutine nodecomponentmergingstatisticsmajorfinalize
subroutine nodecomponentmergingstatisticsnullbuilder
subroutine nodecomponentmergingstatisticsnulldeserializeraw
subroutine nodecomponentmergingstatisticsnullfinalize
subroutine nodecomponentmergingstatisticsrecentbuilder
subroutine nodecomponentmergingstatisticsrecentdeserializeraw
subroutine nodecomponentmergingstatisticsrecentfinalize

subroutine nodecomponentmergingstatisticsstandardbuilder
subroutine nodecomponentmergingstatisticsstandarddeserializeraw
subroutine nodecomponentmergingstatisticsstandardfinalize
subroutine nodecomponentnbodygenericbuilder
subroutine nodecomponentnbodygenericdeserializeraw
subroutine nodecomponentnbodygenericfinalize
subroutine nodecomponentnbodynullbuilder
subroutine nodecomponentnbodynulldeserializeraw
subroutine nodecomponentnbodynullfinalize
subroutine nodecomponentpositionnullbuilder
subroutine nodecomponentpositionnulldeserializeraw
subroutine nodecomponentpositionnullfinalize
subroutine nodecomponentpositionpresetbuilder
subroutine nodecomponentpositionpresetdeserializeraw
subroutine nodecomponentpositionpresetfinalize
subroutine nodecomponentpositionpresetorphansbuilder
subroutine nodecomponentpositionpresetorphansdeserializeraw
subroutine nodecomponentpositionpresetorphansfinalize
subroutine nodecomponentpositiontracedarkmatterbuilder
subroutine nodecomponentpositiontracedarkmatterdeserializeraw
subroutine nodecomponentpositiontracedarkmatterfinalize
subroutine nodecomponentsatellitenullbuilder
subroutine nodecomponentsatellitenulldeserializeraw
subroutine nodecomponentsatellitenullfinalize
subroutine nodecomponentsatelliteorbitingbuilder
subroutine nodecomponentsatelliteorbitingdeserializeraw
subroutine nodecomponentsatelliteorbitingfinalize
subroutine nodecomponentsatellitepresetbuilder
subroutine nodecomponentsatellitepresetdeserializeraw
subroutine nodecomponentsatellitepresetfinalize
subroutine nodecomponentsatellitestandardbuilder
subroutine nodecomponentsatellitestandarddeserializeraw
subroutine nodecomponentsatellitestandardfinalize
subroutine nodecomponentsatelliteverysimplebuilder
subroutine nodecomponentsatelliteverysimpledeserializeraw
subroutine nodecomponentsatelliteverysimplefinalize
subroutine nodecomponentspheroidnullbuilder
subroutine nodecomponentspheroidnulldeserializeraw
subroutine nodecomponentspheroidnullfinalize
subroutine nodecomponentspheroidstandardbuilder
subroutine nodecomponentspheroidstandarddeserializeraw
subroutine nodecomponentspheroidstandardfinalize
subroutine nodecomponentspheroidverysimplebuilder
subroutine nodecomponentspheroidverysimpledeserializeraw

subroutine nodecomponentspheroidverysimplefinalize
subroutine nodecomponentspinnullbuilder
subroutine nodecomponentspinnulldeserializeraw
subroutine nodecomponentspinnullfinalize
subroutine nodecomponentspinpreset3dbuilder
subroutine nodecomponentspinpreset3ddeserializeraw
subroutine nodecomponentspinpreset3dfinalize
subroutine nodecomponentspinpresetbuilder
subroutine nodecomponentspinpresetdeserializeraw
subroutine nodecomponentspinpresetfinalize
subroutine nodecomponentspinrandombuilder
subroutine nodecomponentspinrandomdeserializeraw
subroutine nodecomponentspinrandomfinalize
subroutine nodecomponentspinvitvitskabuilder
subroutine nodecomponentspinvitvitskadeserializeraw
subroutine nodecomponentspinvitvitskafinalize
subroutine positionpresetorphanspositionorphanset
subroutine positionpresetorphanspositionset
subroutine positionpresetorphansvelocityorphanset
subroutine positionpresetorphansvelocityset
subroutine positionpresetpositionset
subroutine positionpresetvelocityset
subroutine positiontracedarkmatterpositionset
subroutine satelliteorbitingpositionset
subroutine satelliteorbitingvelocityset
subroutine spinpreset3dspinvectorgrowthrateset
subroutine spinpreset3dspinvectorset
subroutine spinvitvitskaspinvectorset
subroutine treenodedeserializevaluesfromarray
subroutine treenodeserializevaluestoarray
subroutine node_component_black_hole_standard_output_properties
subroutine node_component_merging_statistics_recent_initialize
subroutine node_component_merging_statistics_recent_thread_initialize
subroutine node_component_spheroid_standard_star_formation_history_rate
subroutine node_component_spheroid_standard_stellar_prprts_history_rate
subroutine stellar_luminosities_create
subroutine stellar_luminosities_expand_filter_set
subroutine stellar_luminosities_initialize
subroutine stellar_luminosities_output
subroutine stellar_luminosities_parameter_map_double
subroutine stellar_luminosities_read_raw
subroutine stellar_luminosities_special_cases
subroutine table_1d_destroy
subroutine table_2d_linlinlin_create
subroutine table_2d_linlinlin_destroy
subroutine table_2dlogloglin_create
subroutine table_2dlogloglin_destroy
subroutine table_generic_1d_create
subroutine table_linear_1d_create
subroutine table_linear_cspline_1d_create

subroutine table_linear_cspline_1d_destroy
subroutine table_linear_monotone_cspline_1d_create
subroutine table_linear_monotone_cspline_1d_destroy
subroutine tensor_r2_d3_sym_destroy
function listconstructorinternal
function listconstructorparameters
function gelmanrubinconstructorinternal
function gelmanrubinisconverged
function intergalacticbackgroundfileconstructorinternal
function intergalacticbackgroundinternalconstructorinternal
subroutine insituscales
subroutine statistics_points_correlation
subroutine statistics_points_power_spectrum
function fileconstructorinternal
subroutine fileread
function nagashima2005constructorinternal
function fileconstructorinternal
subroutine stellar_population_luminosity_tabulate
subroutine stellar_population_luminosity_track
subroutine noninstantaneousscales
subroutine filereadfile
subroutine filetabulation
function filewavelengthinterval
subroutine filewavelengths
function standardinterpolate
subroutine farahifileread
function farahiprobability
subroutine farahiratetabulate
function farahimidpointprobability
subroutine farahimidpointratetabulate
function zhanghuiprobability
function zhanghuihighorderprobability
function cosmicemuvalue
subroutine catalogprojectedcorrelationfunctionperform
function conditionalmassfunctionconstructorinternal
subroutine conditionalmassfunctionperform
subroutine evolveforestsperform
subroutine excursionsetsperform
subroutine halomassfunctionperform
function halomodelprojectedcorrelationfunctionconstructorinternal
subroutine halomodelgenerateperform
subroutine massfunctioncovariancelssangularspectrum
subroutine massfunctioncovariancelsswindowfunction
subroutine massfunctioncovarianceperform
subroutine powerspectraperform
program tests_halo_mass_function_tinker
program test_perfect_hashes
program tests_linear_growth_cosmological_constant
program test_nodes
function intergalacticmediumstateevolveconstructorinternal
subroutine io_hdf5_read_attribute_character_1d_array_allocatable
subroutine io_hdf5_read_attribute_double_1d_array_allocatable
subroutine io_hdf5_read_attribute_integer8_1d_array_allocatable
subroutine io_hdf5_read_attribute_integer8_1d_array_static

subroutine io_hdf5_read_attribute_-integer_1d_array_allocatable	subroutine io_hdf5_read_attribute_-varstring_1d_array_allocatable_do_-read
subroutine io_hdf5_read_dataset_-character_1d_array_allocatable	subroutine io_hdf5_read_dataset_-double_1d_array_allocatable
subroutine io_hdf5_read_dataset_-double_2d_array_allocatable	subroutine io_hdf5_read_dataset_-double_3d_array_allocatable
subroutine io_hdf5_read_dataset_-double_4d_array_allocatable	subroutine io_hdf5_read_dataset_-double_5d_array_allocatable
subroutine io_hdf5_read_dataset_-double_6d_array_allocatable	subroutine io_hdf5_read_dataset_-integer8_1d_array_allocatable
subroutine io_hdf5_read_dataset_-integer8_1d_array_static	subroutine io_hdf5_read_dataset_-integer8_2d_array_allocatable
subroutine io_hdf5_read_dataset_-integer8_2d_array_static	subroutine io_hdf5_read_dataset_-integer_1d_array_allocatable
subroutine io_hdf5_read_dataset_-integer_2d_array_allocatable	subroutine io_hdf5_read_dataset_-varstring_1d_array_allocatable_do_-read
subroutine io_hdf5_read_table_-character_1d_array_allocatable	subroutine io_hdf5_read_table_-integer8_1d_array_allocatable
subroutine io_hdf5_read_table_integer_-1d_array_allocatable	subroutine io_hdf5_read_table_real_1d_-array_allocatable
subroutine io_hdf5_write_attribute_-integer8_1d	subroutine io_hdf5_write_dataset_-integer8_1d
subroutine io_hdf5_write_dataset_-integer8_2d	subroutine xml_array_read_one_column
subroutine xml_array_read_two_column	subroutine xml_list_array_read_one_-column
subroutine mpiinitialize	subroutine hash_perfect_create
subroutine hash_perfect_destroy	subroutine multicounterappend
function multicounterconstructor	subroutine multicounterdestructor
function get_new_assert_result	subroutine unit_tests_finish

subroutine: `add_memory_component`

Description: Add a memory type to the memory reporting strings.
Code lines: 22
Contained by: module memory_management
Modules used: iso_varying_string

interface: `allocatearray`

Description: Generic interface to routines which allocate arrays.
Code lines: 36
Contained by: module memory_management

subroutine: `allocatearray_character_1d`

Description: Allocate a 1D `character` array.
Code lines: 39
Contained by: module memory_management

Modules used: dmemory galacticus_display iso_varying_string string_handling

subroutine: allocatearray_character_1d_kind_int8
Description: Allocate a 1D character array.
Code lines: 39
Contained by: module memory_management
Modules used: dmemory galacticus_display iso_varying_string string_handling

subroutine: allocatearray_complex_c_double_complex_3d
Description: Allocate a 3D complex_c_double_complex array.
Code lines: 39
Contained by: module memory_management
Modules used: dmemory galacticus_display iso_varying_string string_handling

subroutine: allocatearray_complex_c_double_complex_3d_kind_int8
Description: Allocate a 3D complex_c_double_complex array.
Code lines: 39
Contained by: module memory_management
Modules used: dmemory galacticus_display iso_varying_string string_handling

subroutine: allocatearray_double_precision_1d
Description: Allocate a 1D double precision array.
Code lines: 39
Contained by: module memory_management
Modules used: dmemory galacticus_display iso_varying_string string_handling

subroutine: allocatearray_double_precision_1d_kind_int8
Description: Allocate a 1D double precision array.
Code lines: 39
Contained by: module memory_management
Modules used: dmemory galacticus_display iso_varying_string string_handling

subroutine: allocatearray_double_precision_2d
Description: Allocate a 2D double precision array.
Code lines: 39
Contained by: module memory_management
Modules used: dmemory galacticus_display iso_varying_string string_handling

subroutine: allocatearray_double_precision_2d_kind_int8
Description: Allocate a 2D double precision array.
Code lines: 39
Contained by: module memory_management

Modules used: dmemory galacticus_display iso_varying_string string_handling

subroutine: allocatearray_double_precision_3d
Description: Allocate a 3D double precision array.
Code lines: 39
Contained by: module memory_management
Modules used: dmemory galacticus_display iso_varying_string string_handling

subroutine: allocatearray_double_precision_3d_kind_int8
Description: Allocate a 3D double precision array.
Code lines: 39
Contained by: module memory_management
Modules used: dmemory galacticus_display iso_varying_string string_handling

subroutine: allocatearray_double_precision_4d
Description: Allocate a 4D double precision array.
Code lines: 39
Contained by: module memory_management
Modules used: dmemory galacticus_display iso_varying_string string_handling

subroutine: allocatearray_double_precision_4d_kind_int8
Description: Allocate a 4D double precision array.
Code lines: 39
Contained by: module memory_management
Modules used: dmemory galacticus_display iso_varying_string string_handling

subroutine: allocatearray_double_precision_5d
Description: Allocate a 5D double precision array.
Code lines: 39
Contained by: module memory_management
Modules used: dmemory galacticus_display iso_varying_string string_handling

subroutine: allocatearray_double_precision_5d_kind_int8
Description: Allocate a 5D double precision array.
Code lines: 39
Contained by: module memory_management
Modules used: dmemory galacticus_display iso_varying_string string_handling

subroutine: allocatearray_double_precision_6d
Description: Allocate a 6D double precision array.
Code lines: 39
Contained by: module memory_management

Modules used: dmemory galacticus_display iso_varying_string string_handling

subroutine: allocatearray_double_precision_6d_kind_int8
Description: Allocate a 6D double precision array.
Code lines: 39
Contained by: module memory_management
Modules used: dmemory galacticus_display iso_varying_string string_handling

subroutine: allocatearray_integer_1d
Description: Allocate a 1D integer array.
Code lines: 39
Contained by: module memory_management
Modules used: dmemory galacticus_display iso_varying_string string_handling

subroutine: allocatearray_integer_1d_kind_int8
Description: Allocate a 1D integer array.
Code lines: 39
Contained by: module memory_management
Modules used: dmemory galacticus_display iso_varying_string string_handling

subroutine: allocatearray_integer_2d
Description: Allocate a 2D integer array.
Code lines: 39
Contained by: module memory_management
Modules used: dmemory galacticus_display iso_varying_string string_handling

subroutine: allocatearray_integer_2d_kind_int8
Description: Allocate a 2D integer array.
Code lines: 39
Contained by: module memory_management
Modules used: dmemory galacticus_display iso_varying_string string_handling

subroutine: allocatearray_integer_kind_int8_1d
Description: Allocate a 1D integer_kind_int8 array.
Code lines: 39
Contained by: module memory_management
Modules used: dmemory galacticus_display iso_varying_string string_handling

subroutine: allocatearray_integer_kind_int8_1d_kind_int8
Description: Allocate a 1D integer_kind_int8 array.
Code lines: 39
Contained by: module memory_management

Modules used: dmemory galacticus_display iso_varying_string string_handling

subroutine: allocatearray_integer_kind_int8_2d
Description: Allocate a 2D integer_kind_int8 array.
Code lines: 39
Contained by: module memory_management
Modules used: dmemory galacticus_display iso_varying_string string_handling

subroutine: allocatearray_integer_kind_int8_2d_kind_int8
Description: Allocate a 2D integer_kind_int8 array.
Code lines: 39
Contained by: module memory_management
Modules used: dmemory galacticus_display iso_varying_string string_handling

subroutine: allocatearray_logical_1d
Description: Allocate a 1D logical array.
Code lines: 39
Contained by: module memory_management
Modules used: dmemory galacticus_display iso_varying_string string_handling

subroutine: allocatearray_logical_1d_kind_int8
Description: Allocate a 1D logical array.
Code lines: 39
Contained by: module memory_management
Modules used: dmemory galacticus_display iso_varying_string string_handling

subroutine: allocatearray_logical_2d
Description: Allocate a 2D logical array.
Code lines: 39
Contained by: module memory_management
Modules used: dmemory galacticus_display iso_varying_string string_handling

subroutine: allocatearray_logical_2d_kind_int8
Description: Allocate a 2D logical array.
Code lines: 39
Contained by: module memory_management
Modules used: dmemory galacticus_display iso_varying_string string_handling

subroutine: allocatearray_real_1d
Description: Allocate a 1D real array.
Code lines: 39
Contained by: module memory_management

Modules used: dmemory galacticus_display iso_varying_string string_handling

subroutine: allocatearray_real_1d_kind_int8
Description: Allocate a 1D real array.
Code lines: 39
Contained by: module memory_management
Modules used: dmemory galacticus_display iso_varying_string string_handling

subroutine: allocatearray_real_2d
Description: Allocate a 2D real array.
Code lines: 39
Contained by: module memory_management
Modules used: dmemory galacticus_display iso_varying_string string_handling

subroutine: allocatearray_real_2d_kind_int8
Description: Allocate a 2D real array.
Code lines: 39
Contained by: module memory_management
Modules used: dmemory galacticus_display iso_varying_string string_handling

subroutine: allocatearray_real_kind_quad_1d
Description: Allocate a 1D real_kind_quad array.
Code lines: 39
Contained by: module memory_management
Modules used: dmemory galacticus_display iso_varying_string string_handling

subroutine: allocatearray_real_kind_quad_1d_kind_int8
Description: Allocate a 1D real_kind_quad array.
Code lines: 39
Contained by: module memory_management
Modules used: dmemory galacticus_display iso_varying_string string_handling

subroutine: code_memory_usage
Description: Determines the size of the “text” (i.e. code) size.
Code lines: 19
Contained by: module memory_management
Modules used: dmemory

interface: deallocatearray
Description: Generic interface to routines which deallocate arrays.
Code lines: 19
Contained by: module memory_management

subroutine: `deallocatearray_character_1d`
Description: Deallocate a 1D `character` array.
Code lines: 31
Contained by: module `memory_management`
Modules used: `dmemory` `galacticus_display` `iso_varying_string` `string_handling`

subroutine: `deallocatearray_complex_c_double_complex_3d`
Description: Deallocate a 3D `complex_c_double_complex` array.
Code lines: 31
Contained by: module `memory_management`
Modules used: `dmemory` `galacticus_display` `iso_varying_string` `string_handling`

subroutine: `deallocatearray_double_precision_1d`
Description: Deallocate a 1D `double precision` array.
Code lines: 31
Contained by: module `memory_management`
Modules used: `dmemory` `galacticus_display` `iso_varying_string` `string_handling`

subroutine: `deallocatearray_double_precision_2d`
Description: Deallocate a 2D `double precision` array.
Code lines: 31
Contained by: module `memory_management`
Modules used: `dmemory` `galacticus_display` `iso_varying_string` `string_handling`

subroutine: `deallocatearray_double_precision_3d`
Description: Deallocate a 3D `double precision` array.
Code lines: 31
Contained by: module `memory_management`
Modules used: `dmemory` `galacticus_display` `iso_varying_string` `string_handling`

subroutine: `deallocatearray_double_precision_4d`
Description: Deallocate a 4D `double precision` array.
Code lines: 31
Contained by: module `memory_management`
Modules used: `dmemory` `galacticus_display` `iso_varying_string` `string_handling`

subroutine: `deallocatearray_double_precision_5d`
Description: Deallocate a 5D `double precision` array.
Code lines: 31
Contained by: module `memory_management`
Modules used: `dmemory` `galacticus_display`

iso_varying_string string_handling

subroutine: deallocatearray_double_precision_6d
Description: Deallocate a 6D double precision array.
Code lines: 31
Contained by: module memory_management
Modules used: dmemory galacticus_display
iso_varying_string string_handling

subroutine: deallocatearray_integer_1d
Description: Deallocate a 1D integer array.
Code lines: 31
Contained by: module memory_management
Modules used: dmemory galacticus_display
iso_varying_string string_handling

subroutine: deallocatearray_integer_2d
Description: Deallocate a 2D integer array.
Code lines: 31
Contained by: module memory_management
Modules used: dmemory galacticus_display
iso_varying_string string_handling

subroutine: deallocatearray_integer_kind_int8_1d
Description: Deallocate a 1D integer_kind_int8 array.
Code lines: 31
Contained by: module memory_management
Modules used: dmemory galacticus_display
iso_varying_string string_handling

subroutine: deallocatearray_integer_kind_int8_2d
Description: Deallocate a 2D integer_kind_int8 array.
Code lines: 31
Contained by: module memory_management
Modules used: dmemory galacticus_display
iso_varying_string string_handling

subroutine: deallocatearray_logical_1d
Description: Deallocate a 1D logical array.
Code lines: 31
Contained by: module memory_management
Modules used: dmemory galacticus_display
iso_varying_string string_handling

subroutine: deallocatearray_logical_2d
Description: Deallocate a 2D logical array.
Code lines: 31
Contained by: module memory_management

Modules used: dmemory galacticus_display iso_varying_string string_handling

subroutine: deallocatearray_real_1d
Description: Deallocate a 1D real array.
Code lines: 31
Contained by: module memory_management
Modules used: dmemory galacticus_display iso_varying_string string_handling

subroutine: deallocatearray_real_2d
Description: Deallocate a 2D real array.
Code lines: 31
Contained by: module memory_management
Modules used: dmemory galacticus_display iso_varying_string string_handling

subroutine: deallocatearray_real_kind_quad_1d
Description: Deallocate a 1D real_kind_quad array.
Code lines: 31
Contained by: module memory_management
Modules used: dmemory galacticus_display iso_varying_string string_handling

function: memory_usage_get
Code lines: 9
Contained by: module memory_management
Modules used: dmemory

subroutine: memory_usage_record
Description: Record a change in memory usage.
Code lines: 41
Contained by: module memory_management
Modules used: dmemory galacticus_display iso_c_binding iso_varying_string string_handling

subroutine: memory_usage_report
Description: Writes a report on the current memory usage. The total memory use is evaluated and all usages are scaled into convenient units prior to output.
Code lines: 52
Contained by: module memory_management
Modules used: dmemory galacticus_display iso_varying_string

type: memoryusage
Description: Dervied type variable for storing the properties of a single class of memory storage (memory usage, divisor for outputting and suffix for outputting)
Code lines: 6
Contained by: module memory_management

type: `memoryusagelist`
Description: Dervied type variable for storing all memory usage in the code.
Code lines: 3
Contained by: module memory_management

subroutine: `set_memory_prefix`
Description: Given a memory variable, sets the divisor and suffix required to put the memory usage into convenient units for output.
Code lines: 29
Contained by: module memory_management

file: `utility.memory_usage.cpp`

file: `utility.multi_counter.F90`
Description: Contains a module which implements multi-counters - objects which iterate over all combinations of an arbitary number of counters, each with an arbitrary range.
Code lines: 230

module: `multi_counters`
Description: Implements multi-counters - objects which iterate over all combinations of an arbitary number of counters, each with an arbitrary range.
Code lines: 207
Contained by: file utility.multi_counter.F90
Modules used: iso_c_binding
Used by:
- subroutine standardoutput
- module node_property_extractors
- subroutine abundances_output
- subroutine abundances_post_output
- subroutine kepler_orbits_output
- subroutine agestatisticsoutput
- subroutine basicoutput
- subroutine blackholeoutput
- subroutine darkmatterprofileoutput
- subroutine diskoutput
- subroutine dynamicsstatisticsoutput
- subroutine formationtimeoutput
- subroutine hosthistoryoutput
- subroutine hothalooutput
- subroutine indicesoutput
- subroutine interoutputoutput
- subroutine massflowstatisticsoutput
- subroutine mergingstatisticsoutput
- subroutine nbodyoutput
- subroutine node_component_output_null
- subroutine positionoutput
- subroutine satelliteoutput
- subroutine spheroidoutput
- subroutine spinoutput
- subroutine treenodeoutput
- subroutine node_component_black_hole_simple_output
- subroutine node_component_black_hole_standard_output
- subroutine node_component_merging_statistics_recent_output
- subroutine node_component_nbody_generic_output
- subroutine stellar_luminosities_output
- program test_multi_counters

interface: `multicounter`
Description: Constructors for multi-counters.
Code lines: 3
Contained by: module multi_counters

subroutine: `multicounterappend`
Description: Append a new counter with the given `range`.
Code lines: 26
Contained by: module `multi_counters`
Modules used: `galacticus_error` `memory_management`

function: `multicounterconstructor`
Description: Constructor for multi-counters where the ranges are provided.
Code lines: 16
Contained by: module `multi_counters`
Modules used: `galacticus_error` `memory_management`

function: `multicountercount`
Description: Return the number of counters in the multi-counter.
Code lines: 12
Contained by: module `multi_counters`
Modules used: `galacticus_error`

subroutine: `multicounterdestructor`
Description: Destroy a multi-counter object.
Code lines: 8
Contained by: module `multi_counters`
Modules used: `memory_management`

function: `multicounterincrement`
Description: Increment a multi-counter. Return true if increment was possible, false otherwise.
Code lines: 29
Contained by: module `multi_counters`
Modules used: `galacticus_error`

function: `multicounterisfinal`
Description: Return true if a multi-counter is in its final state, false otherwise.
Code lines: 11
Contained by: module `multi_counters`
Modules used: `galacticus_error`

subroutine: `multicounterreset`
Description: Reset the state of the multi-counter.
Code lines: 7
Contained by: module `multi_counters`

function: `multicounterstate`
Description: Return the state of the multi-counter.
Code lines: 14
Contained by: module `multi_counters`
Modules used: `galacticus_error`

file: `utility.regular_expressions.F90`
Description: Contains a module which implements regular expressions by wrappring the GNU C Library implementations.

Code lines: 134

module: `regular_expressions`
Description: Implements regular expressions by wrappring the GNU C Library implementations.
Code lines: 109
Contained by: file utility.regular_expressions.F90
Modules used: iso_c_binding
Used by: subroutine sussingasciiopen, program tests_regular_expressions, subroutine inputparameterscheckparameters

interface: `regex`
Description: Constructor for regular expression object.
Code lines: 3
Contained by: module regular_expressions

function: `regular_expression_constructor`
Description: Constructor for `regEx` objects.
Code lines: 8
Contained by: module regular_expressions

subroutine: `regular_expression_destroy`
Description: Destroy a `regEx` object.
Code lines: 10
Contained by: module regular_expressions

subroutine: `regular_expression_destructor`
Description: Destructor for `regEx` objects.
Code lines: 8
Contained by: module regular_expressions

function: `regular_expression_match`
Description: Returns true if a `regEx` object matches the supplied `string`.
Code lines: 10
Contained by: module regular_expressions

file: `utility.semaphores.F90`
Description: Contains a module which implements semaphores.
Code lines: 200

module: `semaphores`
Description: Implements semaphores.
Code lines: 175
Contained by: file utility.semaphores.F90
Modules used: iso_c_binding
Used by: module galacticus_error, subroutine evolveforestsperform

type: `semaphore`
Code lines: 36

Contained by: module semaphores

subroutine: `semaphore_close`
Code lines: 6
Contained by: module semaphores

function: `semaphore_open`
Code lines: 26
Contained by: module semaphores

subroutine: `semaphore_post`
Code lines: 8
Contained by: module semaphores

subroutine: `semaphore_post_on_error`
Description: Attempts to post to all open semaphores before exiting the code in an error condition.
Code lines: 13
Contained by: module semaphores

subroutine: `semaphore_unlink`
Code lines: 6
Contained by: module semaphores

subroutine: `semaphore_wait`
Code lines: 8
Contained by: module semaphores

type: `semaphorelist`
Code lines: 3
Contained by: module semaphores

file: `utility.stateful_values.F90`
Description: Contains a module of stateful types.
Code lines: 42

module: `stateful_types`
Description: Contains stateful types.
Code lines: 20
Contained by: file utility.stateful_values.F90
Used by: file merger_trees.construct.read.importer.SussingMergerTreeReconstruct.F90 file merger_trees.construct.read.importer.galacticus.F90 file tasks.merger_tree_file_builder.F90

type: `statefuldouble`
Code lines: 3
Contained by: module stateful_types

type: `statefulinteger`
Code lines: 3

Contained by: module stateful_types

type: statefullogical
Code lines: 3
Contained by: module stateful_types

file: utility.string_handling.F90
Description: Contains a module which implements various useful functionality for manipulating character strings.
Code lines: 441

module: string_handling
Description: Implements various useful functionality for manipulating character strings.
Code lines: 419
Contained by: file utility.string_handling.F90
Modules used: iso_varying_string
Used by:

subroutine hopkins2007buildfile	function abundance_pattern_lookup
function atom_lookup	subroutine atomic_data_initialize
subroutine atomiccliecloudytabulate	subroutine atomiccliecloudytabulate
function betaprofileconstructorinternal	function isothermalconstructorinternal
function simpleconstructorinternal	function node_branch_jump
subroutine inter_tree_event_post_evolve	function node_pull_from_tree
function node_push_from_tree	function node_subhalo_promotion
function galactic_structure_radius_-enclosing_density	function galactic_structure_radius_-enclosing_mass
subroutine galactic_structure_radii_-definition_decode	subroutine radiussolve
function galacticus_component_list	subroutine galacticus_output_open_file
function hivshalomassrelationpadmanabhan2017constructorinternal	function luminosityfunctionconstructorinternal
function luminosityfunctionhalphaconstructorinternal	function luminosityfunctionsobral2013hizelsconstructorint
function massfunctionhiconstructorinternal	function massfunctionstellarconstructorinternal
function massfunctionstellarprimusconstructorinternal	function massfunctionstellarukidssudsconstructorinternal
function massfunctionstellarultravistaconstructorinternal	function massfunctionstellarvipersconstructorinternal
function massfunctionstellarzfourgeconstructorinternal	function stellarvshalomassrelationleauthaud2012constructo
subroutine galacticus_build_output	function galacticus_build_string
subroutine galacticus_extra_output_-halo_fourier_profile	function galacticus_version_string
subroutine galacticus_state_retrieve	subroutine galacticus_state_store
function squareconstructorinternal	function squareisinlightcone
function squareposition	function geometrymangleangularpower

function geometrymanglesolidangle
subroutine caputi2011ukidssudsrandomsinitialize
subroutine martin2010alfalfarandomsinitialize
function mangleangularpower
subroutine randompointswindowfunctions
subroutine interface_camb_transfer_function
subroutine interface_fsps_initialize
function buildconstruct
subroutine readassignsplitforestevents
subroutine readbuildisolatedparentpointers
subroutine readbuildsubhalomasshistories
subroutine readcreatenodeindices
subroutine readintertreemergetimeset
subroutine readscanforbranchjumps
subroutine readtimeuntilmergingsubresolution
subroutine sussingasciiopen
subroutine sussinghdf5load
subroutine mergertreestatefromfile
subroutine recordevolutionoutput
subroutine satellitemergerprocess
subroutine standardevolve
subroutine standarderrorhandler
subroutine standardpromote
function augmentbuildtreefromnode
subroutine massaccretionhistoryoperate
subroutine standardmakegroup
function galaxypopulationevaluate
function independentlikelihoodsconstructorparameters
function likelihoodmassfunctiontimeintegrand
subroutine selfboundoperate
function lmnstyemssnlineconstructorinternal
subroutine history_extend
subroutine merger_tree_data_structure_export_galacticus
subroutine windowread
subroutine liwhite2009sdssrandomsinitialize
subroutine fullskywindowfunctions
function manglesolidangle
subroutine filter_response_load
subroutine interface_cloudy_cie_tabulate
subroutine interface_fsps_ssps_tabulate
subroutine readassignmergers
subroutine readbuilddescendentpointers
subroutine readbuildparentpointers
function readconstruct
subroutine readenforcesubhalostatus
subroutine readrootnodeaffinitiesinitial
subroutine readscanformergers
subroutine sussingasciiload
subroutine sussingtreeindicesread
subroutine sussinghdf5open
function nodearrayposition
subroutine evolve_to_time_report
subroutine standarddeadlockoutputtree
function standardtimeevolveto
subroutine standardmerge
subroutine singlelevelhierarchyprocess
subroutine augmentoperate
subroutine merger_tree_structure_output
subroutine standardoutputgroupcreate
function gaussianregressionevaluate
function independentlikelihoodssequantialevaluate
function posteriorasprioriconstructorinternal
subroutine lightconeaddinstances
subroutine odeiv2_solve
subroutine merger_tree_data_structure_export
subroutine merger_tree_data_structure_read_ascii

subroutine merger_tree_data_structure_-read_particles_ascii

subroutine nodecomponentagestatisticsnullserializeascii

subroutine nodecomponentbasiccreate

subroutine nodecomponentbasicnonevolvingserializeascii

subroutine nodecomponentbasicstandardextendedserializeascii

subroutine nodecomponentbasicstandardtrackingserializeascii

subroutine nodecomponentblackholenoncentralserializeascii

subroutine nodecomponentblackholesimpleserializeascii

subroutine nodecomponentdarkmatterprofilecreate

subroutine nodecomponentdarkmatterprofilescalepresetmassserializeascii

subroutine nodecomponentdarkmatterprofilescaleshapeserializeascii

subroutine nodecomponentdisknullserializeascii

subroutine nodecomponentdiskverysimpleserializeascii

subroutine nodecomponentdynamicsstatisticsbarsserializeascii

subroutine nodecomponentdynamicsstatisticsnullserializeascii

subroutine nodecomponentformationtimecreate

subroutine nodecomponentformationtimenullserializeascii

subroutine nodecomponenthosthistorycreate

subroutine nodecomponenthosthistorystandardserializeascii

subroutine nodecomponenthothalocreate

subroutine nodecomponenthothalooutflowtrackingserializeascii

subroutine nodecomponenthothaloverysimpledelayedserializeascii

subroutine nodecomponentagestatisticscreate

subroutine nodecomponentagestatisticsstandardserializeascii

subroutine nodecomponentbasicextendedtrackingserializeascii

subroutine nodecomponentbasicnullserializeascii

subroutine nodecomponentbasicstandardserializeascii

subroutine nodecomponentblackholecreate

subroutine nodecomponentblackholenullserializeascii

subroutine nodecomponentblackholestandardserializeascii

subroutine nodecomponentdarkmatterprofilenullserializeascii

subroutine nodecomponentdarkmatterprofilescaleserializeascii

subroutine nodecomponentdiskcreate

subroutine nodecomponentdiskstandardserializeascii

subroutine nodecomponentdiskverysimplesizeserializeascii

subroutine nodecomponentdynamicsstatisticscreate

subroutine nodecomponentformationtimecole2000serializeascii

subroutine nodecomponentformationtimemassfractionserializeascii

subroutine nodecomponentgeterror

subroutine nodecomponenthosthistorynullserializeascii

subroutine nodecomponenthothalocoldmodeserializeascii

subroutine nodecomponenthothalonullserializeascii

subroutine nodecomponenthothalostandardserializeascii

subroutine nodecomponenthothaloverysimpleserializeascii

subroutine nodecomponentindicescreate

subroutine nodecomponentindicesnullserializeascii

subroutine nodecomponentindicesstandardserializeascii

subroutine nodecomponentinteroutputcreate

subroutine nodecomponentinteroutputnullserializeascii

subroutine nodecomponentinteroutputstandardserializeascii

subroutine nodecomponentmassflowstatisticscreate

subroutine nodecomponentmassflowstatisticsnullserializeascii

subroutine nodecomponentmassflowstatisticsstandardserializeascii

subroutine nodecomponentmergingstatisticscreate

subroutine nodecomponentmergingstatisticsmajorserializeascii

subroutine nodecomponentmergingstatisticsnullserializeascii

subroutine nodecomponentmergingstatisticsrecentserializeascii

subroutine nodecomponentmergingstatisticsstandardserializeascii

subroutine nodecomponentnbodycreate

subroutine nodecomponentnbodygenericserializeascii

subroutine nodecomponentnbodynullserializeascii

subroutine nodecomponentpositioncreate

subroutine nodecomponentpositionnullserializeascii

subroutine nodecomponentpositionpresetorphansserializeascii

subroutine nodecomponentpositionpresetserializeascii

subroutine nodecomponentpositiontracedarkmatterserializeascii

subroutine nodecomponentsatellitecreate

subroutine nodecomponentsatellitenullserializeascii

subroutine nodecomponentsatelliteorbitingserializeascii

subroutine nodecomponentsatellitepresetserializeascii

subroutine nodecomponentsatellitestandardserializeascii

subroutine nodecomponentsatelliteverysimpleserializeascii

subroutine nodecomponentspheroidcreate

subroutine nodecomponentspheroidnullserializeascii

subroutine nodecomponentspheroidstandardserializeascii

subroutine nodecomponentspheroidverysimpleserializeascii

subroutine nodecomponentspincreate

subroutine nodecomponentspinnullserializeascii

subroutine nodecomponentspinpreset3dserializeascii

subroutine nodecomponentspinpresetserializeascii

subroutine nodecomponentspinrandomserializeascii

subroutine nodecomponentspinvitvitskaserializeascii

subroutine tree_node_remove_from_host

subroutine tree_node_remove_from_mergee

subroutine treenodeserializeascii

subroutine treenodeserializexml

subroutine node_component_basic_-standard_promote

subroutine node_component_black_hole_-standard_output_properties

subroutine node_component_disk_-standard_post_step

subroutine node_component_disk_very_-simple_post_evolve

subroutine node_component_disk_very_simple_post_step
subroutine node_component_dynamics_statistics_bars_output
subroutine node_component_formation_time_mass_fraction_node_promotion
subroutine node_component_merging_statistics_major_output
subroutine node_component_nbody_generic_output_names
subroutine node_component_satellite_preset_inter_tree_postprocess
subroutine node_component_satellite_preset_orphanize
subroutine node_component_satellite_preset_satellite_host_change
subroutine node_component_spheroid_standard_post_step
subroutine node_component_spheroid_very_simple_post_step
function stellar_luminosities_index_from_properties
subroutine stellar_luminosities_special_cases
subroutine stellar_luminosities_state_restore
subroutine stellar_luminosities_state_store
function gelmanrubinisconverged
function adaptivegamma
function adaptiveexponent
subroutine annealeddifferentialevolutionupdate
function differentialevolutionconstructorparameters
subroutine differentialevolutionsimulate
function particleswarmconstructorparameters
subroutine particleswarmsimulate
subroutine tempereddifferentialevolutionupdate
subroutine correlationcorrelationlengthcompute
subroutine resumeinitialize
subroutine cole2000get
subroutine covington2008get
subroutine satellite_move_to_new_host
subroutine insituoutput
subroutine metallicitysplitoutput
subroutine statistics_points_power_spectrum
function fileconstructorinternal
subroutine stellar_population_luminosity_tabulate
subroutine catalogprojectedcorrelationfunctionperform
subroutine conditionalmassfunctionperform
subroutine evolveforestsperform
subroutine evolveforestsresumetree
subroutine evolveforestssuspendtree
subroutine halomassfunctionperform
subroutine halomodelgenerateperform
subroutine halospindistributionperform
subroutine intergalacticmediumstateperform
subroutine powerspectraperform
program test_nfw96_concentration_dark_energy
program test_zhao2009_flat
program test_zhao2009_dark_energy
program test_zhao2009_open
program test_locks
program test_string_utilities
subroutine io_hdf5_close
function io_hdf5_datasets
subroutine io_hdf5_write_attribute_varstring_1d
subroutine io_hdf5_write_dataset_varstring_1d
function formatted_date_and_time

function executable_find
function file_name_temporary
function hash_md5
module input_parameters
subroutine inputparameterscheckparameters
subroutine allocatearray_character_1d
subroutine allocatearray_character_1d_kind_int8
subroutine allocatearray_complex_c_double_complex_3d
subroutine allocatearray_complex_c_double_complex_3d_kind_int8
subroutine allocatearray_double_precision_1d
subroutine allocatearray_double_precision_1d_kind_int8
subroutine allocatearray_double_precision_2d
subroutine allocatearray_double_precision_2d_kind_int8
subroutine allocatearray_double_precision_3d
subroutine allocatearray_double_precision_3d_kind_int8
subroutine allocatearray_double_precision_4d
subroutine allocatearray_double_precision_4d_kind_int8
subroutine allocatearray_double_precision_5d
subroutine allocatearray_double_precision_5d_kind_int8
subroutine allocatearray_double_precision_6d
subroutine allocatearray_double_precision_6d_kind_int8
subroutine allocatearray_integer_1d
subroutine allocatearray_integer_1d_kind_int8
subroutine allocatearray_integer_2d
subroutine allocatearray_integer_2d_kind_int8
subroutine allocatearray_integer_kind_int8_1d
subroutine allocatearray_integer_kind_int8_1d_kind_int8
subroutine allocatearray_integer_kind_int8_2d
subroutine allocatearray_integer_kind_int8_2d_kind_int8
subroutine allocatearray_logical_1d
subroutine allocatearray_logical_1d_kind_int8
subroutine allocatearray_logical_2d
subroutine allocatearray_logical_2d_kind_int8
subroutine allocatearray_real_1d
subroutine allocatearray_real_1d_kind_int8
subroutine allocatearray_real_2d
subroutine allocatearray_real_2d_kind_int8
subroutine allocatearray_real_kind_quad_1d
subroutine allocatearray_real_kind_quad_1d_kind_int8
subroutine deallocatearray_character_1d
subroutine deallocatearray_complex_c_double_complex_3d
subroutine deallocatearray_double_precision_1d
subroutine deallocatearray_double_precision_2d
subroutine deallocatearray_double_precision_3d
subroutine deallocatearray_double_precision_4d
subroutine deallocatearray_double_precision_5d

subroutine deallocatearray_double_precision_6d
subroutine deallocatearray_integer_1d
subroutine deallocatearray_integer_2d
subroutine deallocatearray_integer_kind_int8_1d
subroutine deallocatearray_integer_kind_int8_2d
subroutine deallocatearray_logical_1d
subroutine deallocatearray_logical_2d
subroutine deallocatearray_real_1d
subroutine deallocatearray_real_2d
subroutine deallocatearray_real_kind_quad_1d
subroutine memory_usage_record
subroutine unit_tests_finish

interface: `char`
Code lines: 2
Contained by: module string_handling

function: `char_logical`
Description: Convert a logical to a string.
Code lines: 12
Contained by: module string_handling

function: `concatenate_varstr_integer`
Description: Provides a concatenation operator to append an integer number to a `varying_string`.
Code lines: 11
Contained by: module string_handling

function: `concatenate_varstr_integer8`
Description: Provides a concatenation operator to append an integer number to a `varying_string`.
Code lines: 12
Contained by: module string_handling
Modules used: kind_numbers

function: `convert_varstring_to_char`
Description: Convert an array of varying strings into an array of characters.
Code lines: 11
Contained by: module string_handling

interface: `operator(//)`
Code lines: 3
Contained by: module string_handling

function: `string_c_to_fortran`
Description: Convert a C-style character array into a Fortran varying string variable.
Code lines: 13
Contained by: module string_handling
Modules used: `iso_c_binding`

function: `string_count_words`
Description: Return a count of the number of space separated words in `inputString`.
Code lines: 36

Contained by: module string_handling

function: string_join
Description: Joins an array of strings into one long string with the given separator.
Code lines: 14
Contained by: module string_handling

function: string_levenshtein_distance
Description: Compute the Levenshtein distance between strings a and b.
Code lines: 27
Contained by: module string_handling

function: string_lower_case
Description: Converts an input string to lower case.
Code lines: 16
Contained by: module string_handling

function: string_lower_case_first
Description: Converts the first character of an input string to lower case.
Code lines: 13
Contained by: module string_handling

interface: string_split_words
Code lines: 3
Contained by: module string_handling

subroutine: string_split_words_char
Description: Split inputString into words and return as an array.
Code lines: 47
Contained by: module string_handling

subroutine: string_split_words_varstring
Description: Split inputString into words and return as an array.
Code lines: 47
Contained by: module string_handling

function: string_strip
Description: Strips a string of leading and trailing whitespace, including tabs.
Code lines: 26
Contained by: module string_handling

function: string_subscript
Description: Converts an input string to Unicode subscripts.
Code lines: 16
Contained by: module string_handling

function: string_superscript
Description: Converts an input string to Unicode superscripts.
Code lines: 16

Contained by: module string_handling

function: string_upper_case
Description: Converts an input string to upper case.
Code lines: 16
Contained by: module string_handling

function: string_upper_case_first
Description: Converts an input string to upper case.
Code lines: 13
Contained by: module string_handling

file: utility.unit_tests.F90
Description: Contains a module which implements unit testing.
Code lines: 1208

module: unit_tests
Description: Implements unit testing.
Code lines: 1186
Contained by: file utility.unit_tests.F90
Modules used: iso_varying_string
Used by:

program test_diemerkravtsov2014_concentration	program tests_io_hdf5
program tests_io_xml	program test_mpi
program test_nfw96_concentration_dark_energy	program test_ode_solver
program test_ode_solver	program test_prada2011_concentration
program test_zhao2009_flat	program test_zhao2009_dark_energy
program test_zhao2009_open	program test_abundances
program test_accretion_disks	program test_array_monotonicity
program test_black_hole_fundamentals	program tests_bug745815
program tests_comoving_distance	program test_comparison
program test_correa2015_concentration	program test_cooling_functions
program tests_cosmic_age	program test_dark_matter_halo_radius_enclosing_mass
program test_dark_matter_profiles	program test_dark_matter_profiles_generic
program test_dark_matter_profiles_heated	program test_differentiation
program test_gaunt_factors	program test_coordinate_systems
program tests_halo_mass_function_tinker	program test_hashes
program test_hashes_cryptographic	program test_perfect_hashes
program test_initial_mass_functions	program test_integration
program test_interpolation_2d	program test_interpolation
program tests_kepler_orbits	program tests_linear_growth_eds
program tests_linear_growth_cosmological_constant	program tests_linear_growth_dark_energy

program tests_linear_growth_open
program test_locks
program test_make_ranges
program test_correa2015_mah
program test_mass_distributions
program test_math_fast
program test_math_distributions
program test_math_special_functions
program test_meshes
program test_multi_counters
program test_nodes
subroutine test_node_task
program test_parameters
program tests_power_spectrum
program test_random
program tests_regular_expressions
program test_root_finding
program test_search
program tests_sigma
program test_sort
program test_sort_topological
program test_inoue2014
program tests_spherical_collapse_dark_energy_eds
program tests_spherical_collapse_dark_energy_omega_zero_point_six
program tests_spherical_collapse_dark_energy_omega_zero_point_eight
program tests_spherical_collapse_dark_energy_omega_half
program tests_spherical_collapse_dark_energy_omega_two_thirds
program tests_spherical_collapse_dark_energy_lambda
program tests_spherical_collapse_dark_energy_open
program tests_spherical_collapse_flat
program tests_spherical_collapse_nonlinear
program tests_spherical_collapse_open
program test_stellar_populations
program test_stellar_populations_luminosities
program test_string_utilities
program test_tables
program test_tensors
program tests_transfer_functions
program tests_tree_branch_destroy
program test_vectors

interface: `assert`
Description: Generic interface for assert routines.
Code lines: 21
Contained by: module unit_tests

subroutine: `assert_character_1d_array`
Description: Assess and record an assertion about character arguments.
Code lines: 45
Contained by: module unit_tests
Modules used: galacticus_error

subroutine: `assert_character_scalar`
Description: Assess and record an assertion about character arguments.
Code lines: 45
Contained by: module unit_tests
Modules used: galacticus_error

subroutine: `assert_double_1d_array`
Description: Assess and record an assertion about double precision arguments.
Code lines: 68
Contained by: module unit_tests
Modules used: galacticus_error numerical_comparison

subroutine: `assert_double_2d_array`
Description: Assess and record an assertion about double precision arguments.
Code lines: 55
Contained by: module `unit_tests`
Modules used: `galacticus_error` `numerical_comparison`

subroutine: `assert_double_3d_array`
Description: Assess and record an assertion about double precision arguments.
Code lines: 57
Contained by: module `unit_tests`
Modules used: `galacticus_error` `numerical_comparison`

subroutine: `assert_double_4d_array`
Description: Assess and record an assertion about double precision arguments.
Code lines: 59
Contained by: module `unit_tests`
Modules used: `galacticus_error` `numerical_comparison`

subroutine: `assert_double_5d_array`
Description: Assess and record an assertion about double precision arguments.
Code lines: 61
Contained by: module `unit_tests`
Modules used: `galacticus_error` `numerical_comparison`

subroutine: `assert_double_complex_1d_array`
Description: Assess and record an assertion about double complex arguments.
Code lines: 51
Contained by: module `unit_tests`
Modules used: `galacticus_error` `numerical_comparison`

subroutine: `assert_double_scalar`
Description: Assess and record an assertion about double precision arguments.
Code lines: 61
Contained by: module `unit_tests`
Modules used: `galacticus_error` `numerical_comparison`

subroutine: `assert_integer8_1d_array`
Description: Assess and record an assertion about integer arguments.
Code lines: 46
Contained by: module `unit_tests`
Modules used: `galacticus_error` `kind_numbers`

subroutine: `assert_integer8_scalar`
Description: Assess and record an assertion about integer arguments.
Code lines: 46
Contained by: module `unit_tests`
Modules used: `galacticus_error` `kind_numbers`

subroutine: `assert_integer_1d_array`
Description: Assess and record an assertion about integer arguments.
Code lines: 45
Contained by: module `unit_tests`
Modules used: `galacticus_error`

subroutine: `assert_integer_scalar`
Description: Assess and record an assertion about integer arguments.
Code lines: 45
Contained by: module `unit_tests`
Modules used: `galacticus_error`

subroutine: `assert_logical_1d_array`
Description: Assess and record an assertion about integer arguments.
Code lines: 37
Contained by: module `unit_tests`
Modules used: `galacticus_error`

subroutine: `assert_logical_scalar`
Description: Assess and record an assertion about logical arguments.
Code lines: 37
Contained by: module `unit_tests`
Modules used: `galacticus_error`

subroutine: `assert_real_1d_array`
Description: Assess and record an assertion about real arguments.
Code lines: 53
Contained by: module `unit_tests`
Modules used: `galacticus_error` `numerical_comparison`

subroutine: `assert_real_scalar`
Description: Assess and record an assertion about real arguments.
Code lines: 47
Contained by: module `unit_tests`
Modules used: `galacticus_error` `numerical_comparison`

subroutine: `assert_varstring_1d_array`
Description: Assess and record an assertion about character arguments.
Code lines: 45
Contained by: module `unit_tests`
Modules used: `galacticus_error`

subroutine: `assert_varstring_scalar`
Description: Assess and record an assertion about character arguments.
Code lines: 45
Contained by: module `unit_tests`
Modules used: `galacticus_error`

type: `assertresult`
Description: A derived type for storing results of asserts.
Code lines: 6
Contained by: module `unit_tests`

function: `get_new_assert_result`
Description: Get a new assert result object.
Code lines: 23
Contained by: module `unit_tests`
Modules used: `memory_management`

function: `getstatus`
Description: Return the status code for a test on the basis of a boolean pass/fail.
Code lines: 11
Contained by: module `unit_tests`

subroutine: `skip`
Description: Record that a test was skipped.
Code lines: 12
Contained by: module `unit_tests`

subroutine: `unit_tests_begin_group`
Description: Marks that a unit test group has begun.
Code lines: 10
Contained by: module `unit_tests`

subroutine: `unit_tests_end_group`
Description: Marks that a unit test group has ended.
Code lines: 8
Contained by: module `unit_tests`

subroutine: `unit_tests_finish`
Description: Write out the results of unit testing.
Code lines: 60
Contained by: module `unit_tests`
Modules used: `galacticus_display` `memory_management` `string_handling`

Part V.

Contributions and Acknowledgements

19. Contributions

Contributions to the Galacticus project have been made by the following people:

Alex Merson

- hiiRegions.emission_lines.F90
- instruments.filters.F90
- objects.stellar_luminosities.F90
- satellites.merging.timescale.random.F90
- stellar_populations.spectra.F90
- stellar_populations.spectra.file.F90

Andrew Benson

- atomic.ionization_potentials.F90
- atomic.ionization_potentials.Verner.F90
- atomic.rates.recombination.dielectronic.Arnaud85.F90
- atomic.rates.recombination.dielectronic.F90
- black_holes.binaries.initial_separation.Volonteri_2003.F90
- black_holes.binaries.initial_separation.tidal_radius.F90
- black_holes.binaries.recoil_velocity.Campanelli2007.F90
- black_holes.binaries.recoil_velocity.F90
- black_holes.binaries.recoil_velocity.zero.F90
- black_holes.binaries.separation_growth_rate.F90
- black_holes.binaries.separation_growth_rate.standard.F90
- black_holes.binaries.separation_growth_rate.zero.F90
- dark_matter_profiles.structure.concentration.WDM.F90
- dark_matter_profiles_DMO.truncated.exponential.F90
- galactic_structure.potential.F90
- galactic_structure.rotation_curve.gradient.F90
- galactic_structure.velocity_dispersions.F90
- merger_trees.operators.particulate.F90
- objects.nodes.components.black_hole.standard.structure_tasks.F90
- objects.nodes.components.satellite.orbiting.F90
- objects.nodes.components.satellite.orbiting.bound_functions.Inc
- objects.nodes.components.satellite.preset.F90
- objects.tensors.F90

- satellites.dynamical_friction.acceleration.Chandrasekhar1943.F90
- satellites.dynamical_friction.acceleration.F90
- satellites.tidal_heating.rate.F90
- satellites.tidal_heating.rate.Gnedin1999.F90
- satellites.tidal_heating.rate.zero.F90
- satellites.tidal_stripping.rate.F90
- satellites.tidal_stripping.rate.Zentner2005.F90
- satellites.tidal_stripping.rate.zero.F90
- star_formation.rate_surface_density.disks.extended_Schmidt.F90
- structure_formation.cosmological_mass_variance.filtered_power_spectrum.F90
- structure_formation.excursion_sets.first_crossing_distribution.Farahi.F90
- structure_formation.excursion_sets.first_crossing_distribution.Farahi.midpoint.F90
- structure_formation.excursion_sets.first_crossing_distribution.Zhang_Hui_high_-order.F90
- structure_formation.transfer_function.Eisenstein_Hu.F90
- tests.dark_matter_halo_radius_enclosing_mass.F90

Anthony Pullen

- dark_matter_profiles.structure.concentration.WDM.F90
- objects.nodes.components.satellite.orbiting.F90
- objects.nodes.components.satellite.orbiting.bound_functions.Inc
- objects.tensors.F90
- satellites.dynamical_friction.acceleration.Chandrasekhar1943.F90
- satellites.dynamical_friction.acceleration.F90
- satellites.tidal_heating.rate.F90
- satellites.tidal_heating.rate.Gnedin1999.F90
- satellites.tidal_heating.rate.zero.F90
- satellites.tidal_stripping.rate.F90
- satellites.tidal_stripping.rate.Zentner2005.F90
- satellites.tidal_stripping.rate.zero.F90
- structure_formation.transfer_function.Eisenstein_Hu.F90

Arya Farahi

- star_formation.rate_surface_density.disks.extended_Schmidt.F90
- structure_formation.excursion_sets.first_crossing_distribution.Farahi.F90
- structure_formation.excursion_sets.first_crossing_distribution.Zhang_Hui_high_-order.F90

Christoph Behrens

- structure_formation.cosmological_mass_variance.filtered_power_spectrum.F90

- structure_formation.excursion_sets.first_crossing_distribution.Farahi.F90
- structure_formation.excursion_sets.first_crossing_distribution.Farahi.midpoint.F90

Chrsitoph Behrens

- objects.nodes.components.spin.Vitvitska.F90

Daniel McAndrew

- accretion.halo.Naoz_Barkana_2007.F90
- atomic.ionization_potentials.F90
- atomic.ionization_potentials.Verner.F90
- atomic.rates.excitation.collisional.F90
- atomic.rates.excitation.collisional.ScholzWalters91.F90
- atomic.rates.recombination.dielectronic.Arnaud85.F90
- atomic.rates.recombination.dielectronic.F90
- intergalactic_medium.state.internal.F90
- universe.operators.intergalactic_medium_state_evolve.F90

Jianling Gan

- objects.nodes.components.satellite.preset.F90

Luiz Felippe S. Rodrigues

- intergalactic_medium.state.RecFast.F90
- intergalactic_medium.state.file.F90

Markus Haider

- satellites.merging.timescale.infinite.F90

Martin White

- satellites.merging.timescale.Wetzel-White.F90

Omid Sameie

- structure_formation.power_spectrum.variance.window_function.smooth_k_space.F90

Stephanie Dörschner

- objects.merger_tree_data.F90

Stéphane Mangeon

- black_holes.binaries.initial_separation.Volonteri_2003.F90
- black_holes.binaries.initial_separation.tidal_radius.F90
- black_holes.binaries.recoil_velocity.Campanelli2007.F90
- black_holes.binaries.recoil_velocity.F90
- black_holes.binaries.recoil_velocity.zero.F90
- black_holes.binaries.separation_growth_rate.F90
- black_holes.binaries.separation_growth_rate.standard.F90

- black_holes.binaries.separation_growth_rate.zero.F90
- galactic_structure.potential.F90
- galactic_structure.rotation_curve.gradient.F90
- galactic_structure.velocity_dispersions.F90
- objects.nodes.components.black_hole.standard.structure_tasks.F90

Xiaolong Du

- dark_matter_profiles_DMO.truncated.exponential.F90
- merger_trees.operators.particulate.F90
- structure_formation.cosmological_mass_variance.filtered_power_spectrum.F90
- structure_formation.excursion_sets.first_crossing_distribution.Farahi.F90
- structure_formation.excursion_sets.first_crossing_distribution.Farahi.midpoint.F90
- tests.dark_matter_halo_radius_enclosing_mass.F90

20. Acknowledgements

In addition to the tools and libraries required to compile and run Galacticus, development of Galacticus has benefitted from extensive use of the following: GNU Octave, Maxima, Cantor, Kile, Emacs and Valgrind. We are grateful to the members of the GNU Fortran mailing list for invaluable discussions and fixes for compiler problems. We thank John Burkardt for making available the Bivar algorithm for performing interpolation on data irregularly spaced on a 2D plane, Dima Verner for making available codes to compute various atomic data for astrophysics, Warren Perger for making available his pFq code for computing generalized hypergeometric functions, and Shanjie Zhang and Jianming Jin made available code for computing the $E_1(z)$ exponential integral for complex argument. Chris Power provided instructions for installing Galacticus under Mac OS X. The community of Galacticus users[1] have provided invaluable feedback and bug reports. Gian Luigi Granato and Laura Silva kindly provided modifications to their Grasil code to allow it to read Galacticus outputs.

[1] In particular, Christoph Behrens, Jianling Gan, Markus Haider, Harald Höller, Eve Kovacs, Ting-Wen Lan, Adrian Pope, Luiz Felippe Rodrigues, Sergio Sanes, Martin White and Liyan Xu.

Part VI.

Appendices

A. Merger Tree File Format

Galacticus uses a standardized HDF5 file structure for merger tree data. This allows for portability. This format is defined below.

A.1. Basic File Format

Merger trees are stored in HDF5 files for portability and convenience. Additionally, the format is intended to be sufficiently flexible to allow it to desribe merger trees obtained in a wide variety of ways, including Monte Carlo algorithms (e.g. extended Press-Schechter algorithms) and from N-body simulations.

A.1.1. Flexibility and Extensibility

All of the groups/datasets in the file except for the `forestIndex` and `forestHalos` groups are, in principle, optional. This does not mean that a file created without some of these optional groups/datasets will be useable by a given code. It is the responsibility of a given code to check that all data that it requires is present in the file. You are therefore encouraged to include as much information as possible when constructing merger tree files.

Additionally, the file format is intended to be extensible. It is permissible to add additional datasets, for example to describe some other properties of nodes in each tree. Additional datasets should follow the structure of currently defined datasets, i.e. they should be stored as a single dataset combining all trees with nodes listed in the same order as for other datasets. For additional datasets which might be of general use you are encouraged to contact us and recommend them for inclusion in the standard—this allows their name to be standardized.

A.1.2. A Note on Scalar Attributes

Many of the HDF5 attributes discussed in this document are indicated to be scalar (rank 0) attributes. It is allowable within the standard that these be pseudo-scalars (rank 1 arrays contaning a single element). This allows such attributes to be created using the `h5lt` API for example.

A.1.3. Example File Structure

An example of the structure of such a file, called "`example.hdf5`" is shown below using the format of `h5dump`. Each of the groups is described in detail in the following sections.

```
HDF5 "example.hdf5" {
GROUP "/" {
   ATTRIBUTE "formatVersion"
   GROUP "cosmology" {
   }
   GROUP "groupFinder" {
   }
   GROUP "forestHalos" {
   }
   GROUP "forests" {
   }
   GROUP "particles" {
   }
   GROUP "provenance" {
   }
   GROUP "simulation" {
   }
   GROUP "forestIndex" {
   }
```

```
   GROUP "units" {
   }
}
```

A.2. Format Version Attribute

The `formatVersion` attribute specifies which version of the merger tree file format this file follows. The current standard is version 2.

A.3. Cosmology Group

The `cosmology` group describes the cosmological model within which the merger trees contained in the file were constructed. An example of this group, showing standard attributes, is given below.

```
GROUP "cosmology" {
   ATTRIBUTE "HubbleParam" {
      DATATYPE  H5T_IEEE_F64LE
      DATASPACE  SCALAR
      DATA {
      (0): 0.73
      }
   }
   ATTRIBUTE "OmegaMatter" {
      DATATYPE  H5T_IEEE_F64LE
      DATASPACE  SCALAR
      DATA {
      (0): 0.25
      }
   }
   ATTRIBUTE "OmegaLambda" {
      DATATYPE  H5T_IEEE_F64LE
      DATASPACE  SCALAR
      DATA {
      (0): 0.75
      }
   }
   ATTRIBUTE "OmegaBaryon" {
      DATATYPE  H5T_IEEE_F64LE
      DATASPACE  SCALAR
      DATA {
      (0): 0.045
      }
   }
   ATTRIBUTE "powerSpectrumIndex" {
      DATATYPE  H5T_IEEE_F64LE
      DATASPACE  SCALAR
      DATA {
      (0): 1
      }
```

```
   }
   ATTRIBUTE "sigma_8" {
      DATATYPE  H5T_IEEE_F64LE
      DATASPACE  SCALAR
      DATA {
      (0): 0.9
      }
   }
   ATTRIBUTE "transferFunction" {
      DATATYPE  H5T_STRING {
            STRSIZE 7;
            STRPAD H5T_STR_SPACEPAD;
            CSET H5T_CSET_ASCII;
            CTYPE H5T_C_S1;
         }
      DATASPACE  SCALAR
      DATA {
      (0): "CAMB"
      }
   }
}
```

A.3.1. Standard Attributes

The following are standard attributes in the `cosmology` group (others may be added as desired).

`HubbleParam` The Hubble parameter in units of 100 km/s/Mpc at $z = 0$, h_0;

`OmegaMatter` The density of matter (both dark and baryonic matter) in units of the critical density at $z = 0$, $\Omega_{\rm M}$;

`OmegaLambda` The density of dark energy in units of the critical density at $z = 0$, Ω_Λ;

`OmegaBaryon` The density of matter (both dark and baryonic matter) in units of the critical density at $z = 0$, $\Omega_{\rm b}$;

`powerSpectrumIndex` The index of the primordial power spectrum of matter fluctuations, i.e. $n_{\rm s}$ for power spectrum $P(k) \propto k^{n_{\rm s}}$;

`sigma_8` The root-variance of mass fluctuations in real space top-hat spheres of radius $8h^{-1}$Mpc computed from the $z = 0$ linear theory power spectrum, σ_8;

`transferFunction` A descriptor of the transfer function used to compute the power spectrum.

A.4. Group Finder Group

This group, typically relevant only for merger trees derived from N-body simulations, describes the characteristics of the group finding algorithm that was used to find halos in the simulation. An example of this group, showing standard attributes, is given below.

```
GROUP "groupFinder" {
   COMMENT "Group finder parameters."
   ATTRIBUTE "code" {
      DATATYPE  H5T_STRING {
            STRSIZE 7;
            STRPAD H5T_STR_SPACEPAD;
            CSET H5T_CSET_ASCII;
            CTYPE H5T_C_S1;
         }
      DATASPACE  SCALAR
      DATA {
      (0): "SUBFIND"
      }
   }
   ATTRIBUTE "linkingLength" {
      DATATYPE  H5T_IEEE_F64LE
      DATASPACE  SCALAR
      DATA {
      (0): 0.2
      }
   }
   ATTRIBUTE "minimumParticleNumber" {
      DATATYPE  H5T_STD_I32LE
      DATASPACE  SCALAR
      DATA {
      (0): 20
      }
   }
}
```

A.4.1. Standard Attributes

The following are standard attributes in the `groupFinder` group (others may be added as desired).

`code` The name of the group finding code used in the construction of these merger trees;

`linkingLength` For friends-of-friends group finding algorithms the dimensionless (i.e. in units of the mean interparticle spacing) linking length used;

`minimumParticleNumber` The minimum number of particles that a group was required to have in order to be included in a merger tree.

A.5. Simulation Group

This group, typically relevant only for merger trees derived from N-body simulations, describes the characteristics of the simulation from which the trees were derived. An example of this group, showing standard attributes, is given below.

```
 GROUP "simulation" {
   COMMENT "Simulation parameters."
```

```
   ATTRIBUTE "ErrTolIntAccuracy" {
      DATATYPE  H5T_IEEE_F64LE
      DATASPACE  SCALAR
      DATA {
      (0): 0.02
      }
   }
   ATTRIBUTE "TypeOfTimestepCriterion" {
      DATATYPE  H5T_STD_I32LE
      DATASPACE  SCALAR
      DATA {
      (0): 0
      }
   }
   ATTRIBUTE "boxSize" {
      DATATYPE  H5T_IEEE_F64LE
      DATASPACE  SCALAR
      DATA {
      (0): 500
      }
   }
   ATTRIBUTE "code" {
      DATATYPE  H5T_STRING {
            STRSIZE 8;
            STRPAD H5T_STR_SPACEPAD;
            CSET H5T_CSET_ASCII;
            CTYPE H5T_C_S1;
         }
      DATASPACE  SCALAR
      DATA {
      (0): "GADGET-2"
      }
   }
   ATTRIBUTE "initialConditions" {
      DATATYPE  H5T_STRING {
            STRSIZE 5;
            STRPAD H5T_STR_SPACEPAD;
            CSET H5T_CSET_ASCII;
            CTYPE H5T_C_S1;
         }
      DATASPACE  SCALAR
      DATA {
      (0): "glass"
      }
   }
   ATTRIBUTE "softeningKernel" {
      DATATYPE  H5T_STRING {
            STRSIZE 6;
            STRPAD H5T_STR_SPACEPAD;
```

```
            CSET H5T_CSET_ASCII;
            CTYPE H5T_C_S1;
         }
      DATASPACE  SCALAR
      DATA {
      (0): "spline"
      }
   }
   ATTRIBUTE "softeningPlummerEquivalent" {
      DATATYPE  H5T_IEEE_F64LE
      DATASPACE  SCALAR
      DATA {
      (0): 0.005
      }
   }
   ATTRIBUTE "startRedshift" {
      DATATYPE  H5T_IEEE_F64LE
      DATASPACE  SCALAR
      DATA {
      (0): 127
      }
   }
}
```

A.5.1. Standard Attributes

The following are standard attributes in the `simulation` group (others may be added as desired).

`boxSize` Relevant for cubic volumes typical of cosmological simulations, this attributes gives the length of the box in whatever unit system the file used (see §A.6);

`code` The name of the code used to run the simulation;

`initialConditions` A description of the initial conditions;

`softeningKernel` A description of the softening kernel used;

`softeningPlummerEquivalent` The equivalent Plummer softening length;

`startRedshift` The redshift at which the simulation was begun.

GADGET-specific Standard Attributes

The following are standard attributes in the `simulation` group specifically relevant to simulations run with the GADGET code. They typically reflect the values of parameters used by that code.

`ErrTolIntAccuracy` The integration accuracy used by GADGET;

`TypeOfTimestepCriterion` The type of timestepping criterion used by GADGET;

`SofteningGas` Specifies the (comoving) softening of the first particle group in GADGET;

`SofteningHalo` Specifies the (comoving) softening of the second particle group in GADGET;

`SofteningDisk` Specifies the (comoving) softening of the third particle group in GADGET;

`SofteningBulge` Specifies the (comoving) softening of the fourth particle group in GADGET;

`SofteningStars` Specifies the (comoving) softening of the fifth particle group in GADGET;

`SofteningBndry` Specifies the (comoving) softening of the sixth particle group in GADGET;

`SofteningGasMaxPhys` Specifies the maximum physical softening of the first particle group in GADGET;

`SofteningHaloMaxPhys` Specifies the maximum physical softening of the second particle group in GADGET;

`SofteningDiskMaxPhys` Specifies the maximum physical softening of the third particle group in GADGET;

`SofteningBulgeMaxPhys` Specifies the maximum physical softening of the fourth particle group in GADGET;

`SofteningStarsMaxPhys` Specifies the maximum physical softening of the fifth particle group in GADGET;

`SofteningBndryMaxPhys` Specifies the maximum physical softening of the sixth particle group in GADGET.

A.6. Units Group

This group describes the unit system used throughout the file. Attributes should be included for length, mass and velocity units. In each case, three attributes are required to describe the units used (in the following `quantity` refers to `length`, `mass`, `time` or `velocity`):

`quantityUnitsInSI` The units of this quantity expressed in the SI system;

`quantityHubbleExponent` The exponent of the reduced Hubble constant, h, appearing in the units for this quantity;

`quantityScaleFactorExponent` The exponent, n, of the expansion factor, a, required to convert this quantity into physical units. That is, multiplying this quantity by a^n will give the quantity in physical units.

For example, if lengths in the file are expressed in units of comoving h^{-1} Mpc, then we would have

```
lengthUnitsInSI           =  3.08568e+22
lengthHubbleExponent      = -1
lengthScaleFactorExponent =  1
```

This allows a code reading the data from a merger tree file to automatically convert it into whatever unit/coordinate system it chooses.

An example of this group, showing standard attributes, is given below.

```
ATTRIBUTE "lengthHubbleExponent" {
   DATATYPE  H5T_STD_I32LE
   DATASPACE  SCALAR
   DATA {
   (0): -1
   }
```

```
}
ATTRIBUTE "lengthScaleFactorExponent" {
   DATATYPE  H5T_STD_I32LE
   DATASPACE  SCALAR
   DATA {
   (0): 1
   }
}
ATTRIBUTE "lengthUnitsInSI" {
   DATATYPE  H5T_IEEE_F64LE
   DATASPACE  SCALAR
   DATA {
   (0): 3.08568e+22
   }
}
ATTRIBUTE "massHubbleExponent" {
   DATATYPE  H5T_STD_I32LE
   DATASPACE  SCALAR
   DATA {
   (0): -1
   }
}
ATTRIBUTE "massScaleFactorExponent" {
   DATATYPE  H5T_STD_I32LE
   DATASPACE  SCALAR
   DATA {
   (0): 0
   }
}
ATTRIBUTE "massUnitsInSI" {
   DATATYPE  H5T_IEEE_F64LE
   DATASPACE  SCALAR
   DATA {
   (0): 1.98892e+40
   }
}
ATTRIBUTE "timeHubbleExponent" {
   DATATYPE  H5T_STD_I32LE
   DATASPACE  SCALAR
   DATA {
   (0): 0
   }
}
ATTRIBUTE "timeScaleFactorExponent" {
   DATATYPE  H5T_STD_I32LE
   DATASPACE  SCALAR
   DATA {
   (0): 0
   }
```

```
   }
   ATTRIBUTE "timeUnitsInSI" {
      DATATYPE  H5T_IEEE_F64LE
      DATASPACE  SCALAR
      DATA {
      (0): 3.1556926e+16
      }
   }
}   ATTRIBUTE "velocityHubbleExponent" {
      DATATYPE  H5T_STD_I32LE
      DATASPACE  SCALAR
      DATA {
      (0): 0
      }
   }
   ATTRIBUTE "velocityScaleFactorExponent" {
      DATATYPE  H5T_STD_I32LE
      DATASPACE  SCALAR
      DATA {
      (0): 0
      }
   }
   ATTRIBUTE "velocityUnitsInSI" {
      DATATYPE  H5T_IEEE_F64LE
      DATASPACE  SCALAR
      DATA {
      (0): 1000
      }
   }
}
```

A.7. Forest Halos Group

The `forestHalos` group contains the data describing the actual merger tree forests. Nodes from each forest must be stored contiguously. An example of this group is given below. In this example, <`nodeCount`> is the total number of nodes in all merger tree forests.

```
GROUP "forestHalos" {
   ATTRIBUTE "haloMassesIncludeSubhalos" {
      DATATYPE  H5T_STD_I32LE
      DATASPACE  SCALAR
      DATA {
      (0): 0
      }
   }
   ATTRIBUTE "haloAngularMomentaIncludeSubhalos" {
      DATATYPE  H5T_STD_I32LE
      DATASPACE  SCALAR
      DATA {
```

```
   (0): 0
   }
}
ATTRIBUTE "forestsAreSelfContained" {
   DATATYPE  H5T_STD_I32LE
   DATASPACE  SCALAR
   DATA {
   (0): 1
   }
}
ATTRIBUTE "treesHaveSubhalos" {
   DATATYPE  H5T_STD_I32LE
   DATASPACE  SCALAR
   DATA {
   (0): 1
   }
}
ATTRIBUTE "velocitiesIncludeHubbleFlow" {
   DATATYPE  H5T_STD_I32LE
   DATASPACE  SCALAR
   DATA {
   (0): 0
   }
}
ATTRIBUTE "positionsArePeriodic" {
   DATATYPE  H5T_STD_I32LE
   DATASPACE  SCALAR
   DATA {
   (0): 0
   }
}
DATASET "descendentIndex" {
COMMENT "The index of each descendent node."
   DATATYPE  H5T_STD_I64LE
   DATASPACE  SIMPLE { ( <nodeCount> ) / ( <nodeCount> ) }
}
DATASET "expansionFactor" {
COMMENT "The expansion factor of each node."
   DATATYPE  H5T_IEEE_F64LE
   DATASPACE  SIMPLE { ( <nodeCount> ) / ( <nodeCount> ) }
}
DATASET "halfMassRadius" {
COMMENT "The half mass radius of each node."
   DATATYPE  H5T_IEEE_F64LE
   DATASPACE  SIMPLE { ( <nodeCount> ) / ( <nodeCount> ) }
}
DATASET "hostIndex" {
COMMENT "The index of each host node."
   DATATYPE  H5T_STD_I64LE
```

```
   DATASPACE  SIMPLE { ( <nodeCount> ) / ( <nodeCount> ) }
}
DATASET "nodeIndex" {
COMMENT "The index of each node."
   DATATYPE  H5T_STD_I64LE
   DATASPACE  SIMPLE { ( <nodeCount> ) / ( <nodeCount> ) }
}
DATASET "nodeMass" {
COMMENT "The mass of each node."
   DATATYPE  H5T_IEEE_F64LE
   DATASPACE  SIMPLE { ( <nodeCount> ) / ( <nodeCount> ) }
}
DATASET "particleCount" {
COMMENT "The number of entries within the particles group for this node."
   DATATYPE  H5T_STD_I64LE
   DATASPACE  SIMPLE { ( <nodeCount> ) / ( <nodeCount> ) }
}
DATASET "particleStart" {
COMMENT "The index within the particles group at which the particle data for this node is stored."
   DATATYPE  H5T_STD_I64LE
   DATASPACE  SIMPLE { ( <nodeCount> ) / ( <nodeCount> ) }
}
DATASET "position" {
COMMENT "The position of each node."
   DATATYPE  H5T_IEEE_F64LE
   DATASPACE  SIMPLE { ( <nodeCount>, 3 ) / ( <nodeCount>, 3 ) }
}
DATASET "scaleRadius" {
COMMENT "The scale radius of each node."
   DATATYPE  H5T_IEEE_F64LE
   DATASPACE  SIMPLE { ( <nodeCount> ) / ( <nodeCount> ) }
}
DATASET "redshift" {
COMMENT "The redshift of each node."
   DATATYPE  H5T_IEEE_F64LE
   DATASPACE  SIMPLE { ( <nodeCount> ) / ( <nodeCount> ) }
}
DATASET "spin" {
COMMENT "The spin of each node."
   DATATYPE  H5T_IEEE_F64LE
   DATASPACE  SIMPLE { ( <nodeCount>, 3 ) / ( <nodeCount>, 3 ) }
}
DATASET "time" {
COMMENT "The time of each node."
   DATATYPE  H5T_IEEE_F64LE
   DATASPACE  SIMPLE { ( <nodeCount> ) / ( <nodeCount> ) }
}
DATASET "velocity" {
COMMENT "The velocity of each node."
```

```
      DATATYPE  H5T_IEEE_F64LE
      DATASPACE  SIMPLE { ( <nodeCount>, 3 ) / ( <nodeCount>, 3 ) }
   }
}
```

A.7.1. Standard Attributes

The following are standard attributes in the `forestHalos` group (others may be added as desired).

`haloMassesIncludeSubhalos` Indicates whether or not the masses of halos include the masses of any subhalos that they may contain. A value of 0 implies that halo masses do not include masses of subhalos, while a value of 1 indicates that they do;

`haloAngularMomentaIncludeSubhalos` Indicates whether or not the angular momenta of halos include the angular momenta contributed by any subhalos that they may contain. A value of 0 implies that halo masses do not include the contribution of subhalos, while a value of 1 indicates that they do. If this attribute is not present it is assumed to be equal to the `haloMassesIncludeSubhalos` attribute;

`forestsAreSelfContained` Indicates whether or not forests are self-contained, in the sense that nodes never transfer from one forest to another. A value of 0 implies that nodes *can* move from one tree to another, while a value of 1 implies that they can not;

`treesHaveSubhalos` Indicates whether or not trees contain information on subhalos. A value of 0 implies that they do not, while a value of 1 implies that they do. This attribute is a convenience, as subhalo presence can be determined from the node data directly;

`velocitiesIncludeHubbleFlow` Indicates whether or not velocities include the Hubble flow. A value of 0 indicates that they do not, while a value of 1 indicates that they do. See §10.3 for important notes on velocity definitions in Galacticus.

`positionsArePeriodic` Indicates whether or not positions are periodic (as in a cosmological cube simulation). A value of 0 indicates that they are not, while a value of 1 indicates that they are periodic, with a period of `boxSize`.

A.7.2. Standard Datasets

The following are standard datasets in the `forestHalos` group.

`angularMomentum` The angular momentum of the halo. This can be either the magnitude of the angular momentum or a 3-D vector;

`expansionFactor` The expansion factor (normalized to unity at the present day) at which this node is identified (note that only one of the `expansionFactor`, `redshift` and `time` datasets is required, since they are simply related, but multiple *can* be present);

`descendentIndex` The `nodeIndex` of the descendent of this node in the merger tree, or -1 if there is no descendent;

`halfMassRadius` The radius containing half the mass of the node;

`hostIndex` The `nodeIndex` of the node which hosts this node. For nodes that are self-hosting (i.e. that are not subhalos inside another halo), the value of `hostIndex` should be set equal to the node's own `nodeIndex`;

`nodeIndex` An ID number for the node, unique at least within each tree. If nodes are able to move from one tree to another, the ID must be unique within all trees. No other constraints are placed on `nodeIndex` (e.g. it *does not* have to be monotonically increasing within the file for example);

`nodeMass` The mass of the node;

`position` The three dimensional position of this node;

`redshift` The redshift at which this node is identified (note that only one of the `expansionFactor`, `redshift` and `time` datasets is required, since they are simply related, but multiple *can* be present);

`scaleRadius` The characteristic scale radius in the node (typically, but not necessarily, the NFW scale radius);

`spin` The spin parameter, λ, of the halo. This can be either the spin magnitude or a 3-D vector;

`time` The time at which this node is identified (note that only one of the `expansionFactor`, `redshift` and `time` datasets is required, since they are simply related, but multiple *can* be present);

`velocity` The three dimensional velocity of the node. See §10.3 for important notes on velocity definitions in Galacticus;

`particleIndexStart` If the `particles` group is included, this dataset should give the index of the first entry in the particle datasets that corresponds to the particle associated with this node;

`particleIndexCount` If the `particles` group is included, this dataset should give the number of entries in particle datasets that correspond to the particle associated with this node;

Note that, internally, Galacticus uses a different naming convention for the links between nodes. Specifically, an isolated node's descendent is known as its parent, while a satellite node's host is also referred to as the parent. By default, Galacticus will output a `parentIndex` dataset, which therefore specifies either the descendent or host, depending on whether the node in question is isolated or not. To additionally output information which matches the use of "descendent" and "host" in these merger tree files, set both of the input parameters `[outputDescendentIndices]` and `[outputHostIndices]` to `true`. This will result in the output of two additional datasets, `descendentIndex` and `hostIndex` which correspond to the definitions used in the merger tree file.

A.8. Forest Index Group

The `forestIndex` group contains indexing information which describes which sections of the datasets in the `forestHalos` group belong to each forest. An example of this group is given below.

```
GROUP "forestIndex" {
   DATASET "firstNode" {
   COMMENT "Position of the first node in each forest in the forestHalos datasets."
      DATATYPE  H5T_STD_I32LE
      DATASPACE  SIMPLE { ( <forestCount> ) / ( <forestCount> ) }
   }
   DATASET "numberOfNodes" {
   COMMENT "Number of nodes in each forst."
      DATATYPE  H5T_STD_I32LE
      DATASPACE  SIMPLE { ( <forestCount> ) / ( <forestCount> ) }
   }
```

```
   DATASET "forestIndex" {
   COMMENT "Unique index of forst."
      DATATYPE  H5T_STD_I64LE
      DATASPACE  SIMPLE { ( <forestCount> ) / ( <forestCount> ) }
   }
   DATASET "forestWeight" {
   COMMENT "The number of such forests required per unit volume to create a representative sample."
      DATATYPE  H5T_STD_F64LE
      DATASPACE  SIMPLE { ( <forestCount> ) / ( <forestCount> ) }
   }
}
```

A.8.1. Standard Datasets

`firstNode` For each forest, gives the position[1] in the `forestHalos` datasets of the first node in the forest (note that dataset indexing begins at 0). This *does not* necessarily have to be a root node of the forest—nodes in a single forest can be stored in any order in the `forestHalos` datasets, providing that they are contiguous;

`numberOfNodes` For each forest, gives the number of nodes in the forest;

`forestIndex` A unique ID number for each forest;

`forestWeight` A weight factor specifying the number density of each forest required to construct a representative sample. If not present, it is acceptable to assume that the weight is $1/\texttt{boxSize}^3$ if that attribute is present in the `simulation` group.

A.9. Forests Group

The `forests` group is optional and provides a convenience method for accessing the properties of individual forests. If present, it contains one group for each tree in the file, named `forest<forestID>` where `<forestID>` is the ID of the forest. Each of these groups should contain a set of scalar references to the sections of the datasets in the `forestHalos` group to which this forest corresponds. For example, the `descendentIndex` reference for the forest with ID number 89 would be as follows:

```
GROUP "forests/forest89" {
   DATASET "descendentIndex" {
      DATATYPE  H5T_REFERENCE
      DATASPACE  SCALAR
      DATA {
         DATASET /forestHalos/descendentIndex {(<indexBegin>)-(<indexEnd>)}
      }
   }
}
```

[1] That is, it gives the array index of the first node of the forest in the `forestHalos/nodeIndex` dataset for example. It does *not* give the `nodeIndex` of the first node in the forest.

A.10. Particles Group

The `particles` group is optional and contains information on particle trajectories. It is intended for use with merger trees derived from N-body simulations for which it is often useful to track the location of, for example, the most bound particle associated with a subhalo even after that subhalo can no longer be tracked in the simulation. An example of this group is given below. In this example, $< particleCount >$ is the total number of particles included in the group.

```
GROUP "particles" {
   DATASET "particleID" {
   COMMENT "The ID of each particle."
      DATATYPE  H5T_STD_I64LE
      DATASPACE  SIMPLE { ( <particleCount> ) / ( <particleCount> ) }
   }
   DATASET "redshift" {
   COMMENT "The redshift of each particle."
      DATATYPE  H5T_IEEE_F64LE
      DATASPACE  SIMPLE { ( <particleCount> ) / ( <particleCount> ) }
   }
   DATASET "time" {
   COMMENT "The time of each particle."
      DATATYPE  H5T_IEEE_F64LE
      DATASPACE  SIMPLE { ( <particleCount> ) / ( <particleCount> ) }
   }
   DATASET "expansionFactor" {
   COMMENT "The expansion factor of each particle."
      DATATYPE  H5T_IEEE_F64LE
      DATASPACE  SIMPLE { ( <particleCount> ) / ( <particleCount> ) }
   }
   DATASET "position" {
   COMMENT "The position of each node."
      DATATYPE  H5T_IEEE_F64LE
      DATASPACE  SIMPLE { ( <particleCount>, 3 ) / ( <particleCount>, 3 ) }
   }
   DATASET "velocity" {
   COMMENT "The velocity of each node."
      DATATYPE  H5T_IEEE_F64LE
      DATASPACE  SIMPLE { ( <particleCount>, 3 ) / ( <particleCount>, 3 ) }
   }
}
```

Each particle should be stored contiguously (i.e. entries with the same `particleID` should be consecutive) and it is frequently convenient (although not required) that entries for each particle be arranged in order of increasing cosmic time.

A.10.1. Standard Datasets

`particleID` An ID, unique within the entire simulation, for each particle;

`redshift` The redshift at which the particle is recorded (a single particle can appear in these datasets at multiple times);

`time` The time at which the particle is recorded (a single particle can appear in these datasets multiple times at different redshifts);

`expansionFactor` The expansion factor at which the particle is recorded (a single particle can appear in these datasets multiple times at different expansion factors);

`position` The spatial position of the particle;

`velocity` The velocity of the particle. See §10.3 for important notes on velocity definitions in GALACTICUS.

Note that only one of the `expansionFactor`, `time` and `redshift` datasets is required, as they are simply related, but multiple of them *can* be present.

A.11. Merger Tree Builder

Galacticus contains software which builds merger tree files in the format described in §A.1 from merger tree descriptions in other formats, such as ASCII output from an SQL database. The merger tree building engine can be found in `source/objects.merger_tree_data.F90`. Examples of how this engine is used can be found in `source/Millennium_Merger_Tree_File_Maker.F90` and `source/merger_trees.file_maker.Millennium.F90` which are designed to work with the `scripts/aux/Millennium_Trees_Grab.pl` script to convert data extracted from the Millennium Simulation database into a format that Galacticus can read, and in `source/Simple_Merger_Tree_File_Maker.F90` and `source/merger_trees.file_maker.simple.F90` which are designed to work with ASCII file representations of mergers trees that contain just the mass, redshift and descendent of each node.

The basic process for building a merger tree file is to inform the engine of the data file to read and where specific information is located within that file. The data can then be processed and, finally, output in the required format. Specific interfaces that can be used are described below. Many of these interfaces work on an object `mergerTrees` of `mergerTreeData` type. This object stores all information on the merger trees while they are being internally processed.

Setting property locations: Before reading data from a file it is necessary to inform the tree builder engine of which column in the file corresponds to which property. This is done with repeated calls to `setProperty`, one for each column to read, as follows:

```
call mergerTrees%setProperty(propertyType,columnIndex)
```

where `columnIndex` is the number of the column (counting from 1) which contains the property specified by `propertyType`. `propertyType` can take one of the following values:

`propertyTypeTreeIndex` A unique ID number for the tree to which this node belongs;

`propertyTypeNodeIndex` An ID (unique within the tree) for this node;

`propertyTypeDescendentIndex` The ID of the node's descendent node;

`propertyTypeHostIndex` The ID of the larger halo in which this node is hosted (equal to the node's own ID if the node is self-hosting);

`propertyTypeRedshift` The redshift of the node;

`propertyTypeNodeMass` The mass of the node;

`propertyTypeParticleCount` The number of particles in the node;

`propertyTypePositionX` The x-position of the node (if present, both y and z components must also be present);

`propertyTypePositionY` The y-position of the node (if present, both x and z components must also be present);

`propertyTypePositionZ` The z-position of the node (if present, both x and y components must also be present);

`propertyTypeVelocityX` The x-velocity of the node (if present, both y and z components must also be present);

`propertyTypeVelocityY` The y-velocity of the node (if present, both x and z components must also be present);

`propertyTypeVelocityZ` The z-velocity of the node (if present, both x and y components must also be present);

`propertyTypeSpinX` The x component of the node's spin parameter (if present, both y and z components must also be present; cannot be present if spin magnitude is given);

`propertyTypeSpinY` The y component of the node's spin parameter (if present, both x and z components must also be present; cannot be present if spin magnitude is given);

`propertyTypeSpinZ` The z component of the node's spin parameter (if present, both x and y components must also be present; cannot be present if spin magnitude is given);

`propertyTypeSpin` The magnitude of the node's spin parameter (cannot be present if spin vector components are given);

`propertyTypeAngularMomentumX` The x-component of the node's angular momentum (if present, both y and z components must also be present; cannot be present if angular momentum magnitude is given);

`propertyTypeAngularMomentumY` The y-component of the node's angular momentum (if present, both x and z components must also be present; cannot be present if angular momentum magnitude is given);

`propertyTypeAngularMomentumZ` The z-component of the node's angular momentum (if present, both x and y components must also be present; cannot be present if angular momentum magnitude is given);

`propertyTypeAngularMomentum` The magnitude of the node's angular momentum (cannot be present if angular momentum vector components are given);

`propertyTypeHalfMassRadius` The half-mass radius of the node;

`propertyTypeMostBoundParticleIndex` The index of the most bound particle in this node.

Not all properties must be specified—any required properties that are not specified will result in an error. Likewise, some properties, if present, require that other properties also be present. For example, if any of the position properties is given then all three positions are required.

Reading ASCII data: Once property columns have been specified, data from an ASCII file with one node per line can be read as follows:

```
call mergerTrees%readASCII(nodesFile,lineNumberStart,lineNumberStop,separator=",")
```

where `nodesFile` is the name of the file to read. The optional `lineNumberStart` and `lineNumberEnd` arguments give the first and last lines of the file to read, while the optional `separator` argument specifies the character used to separate columns (white space is assumed by default).

Setting particle property locations: If particle information is to be stored in the file, the locations of particle properties within the input file must be specified with repeated calls to `setParticleProperty` as follows:

```
call mergerTrees%setParticleProperty(propertyType,columnIndex)
```

where `columnIndex` is the number of the column (counting from 1) which contains the property specified by `propertyType`. `propertyType` can take one of the following values:

`propertyTypeParticleIndex` A unique ID for the particle;

`propertyTypeRedshift` The redshift of the particle;

`propertyTypeNodeMass` The mass of the particle;

`propertyTypeParticleCount` The number of particles in the particle;

`propertyTypePositionX` The x-position of the particle (if present, both y and z components must also be present);

`propertyTypePositionY` The y-position of the particle (if present, both x and z components must also be present);

`propertyTypePositionZ` The z-position of the particle (if present, both x and y components must also be present);

`propertyTypeVelocityX` The x-velocity of the particle (if present, both y and z components must also be present);

`propertyTypeVelocityY` The y-velocity of the particle (if present, both x and z components must also be present);

`propertyTypeVelocityZ` The z-velocity of the particle (if present, both x and y components must also be present).

Reading ASCII particle data: Once property columns have been specified, particle data from an ASCII file with one particle per line can be read as follows:

```
call mergerTrees%readParticlesASCII(particlesFile,lineNumberStart,lineNumberStop,separator=",")
```

where `particlesFile` is the name of the file to read. The optional `lineNumberStart` and `lineNumberEnd` arguments give the first and last lines of the file to read, while the optional `separator` argument specifies the character used to separate columns (white space is assumed by default).

Setting particle mass: The particle mass, `particleMass`, can be specified using:

```
call mergerTrees%setParticleMass(particleMass)
```

Specifying tree self-containment: Whether or not trees are self-contained can be specified using:

```
call mergerTrees%setSelfContained([.true.|.false.])
```

Specifying Hubble flow inclusion: Whether or not velocities include the Hubble flow can be speified using:

```
call mergerTrees%setIncludesHubbleFlow([.true.|.false.])
```

Specifying subhalo mass inclusion: Whether or not halo masses include the masses of any subhalos can be specified using:

```
call mergerTrees%setIncludesSubhaloMasses([.true.|.false.])
```

Specifying reference creation: Whether or not HDF5 reference to individual merger trees within the `haloTrees` datasets should be made can be specified using:

```
call mergerTrees%makeReferences([.true.|.false.])
```

Specifying units: The units used in the files can be specified with repeated calls to `setUnits` as follows:

```
call mergerTrees%setUnits(unitsType,unitsInSI,hubbleExponent,scaleFactorExponent)
```

where `unitsType` is one of:

`unitsMass` Units of mass;

`unitsLength` Units of length;

`unitsTime` Units of time;

`unitsVelocity` Units of velocity;

`unitsInSI` gives the units in the SI system, `hubbleExponent` specifies the power to which h appears in the units and `scaleFactorExponent` specifies the number of powers of the expansion factor by which the quantity should be multiplied to place it into physical units.

Adding metadata: Meta-data can be added to the file by making repeated calls to `addMetadata` as follows:

```
call mergerTrees%addMetadata(metaDataType,label,value)
```

where `metaDataType` is one of:

`metaDataGeneric` Add to the generic `metaData` group;

`metaDataCosmology` Add to the `cosmology` group;

`metaDataSimulation` Add to the `simulation` group;

`metaDataGroupFinder` Add to the `groupFinder` group;

`metaDataTreeBuilder` Add to the `treeBuilder` group;

`metaDataProvenance` Add to the `provenance` group,

`label` is a label for this metadatum and `value` is the value to store. Currently integer, double precision and character data types are supported for metadata.

Exporting the data: Once the data has been read, units and properties specified and any metadata added, the trees can be exported to an HDF5 file using:

```
call mergerTrees%export(outputFile,outputFormat,hdfChunkSize,hdfCompressionLevel)
```

where `outputFile` is the name of the file to which the trees should be exported, `outputFormat` specifies the format to use (see §A.11.1), and `hdfChunkSize` and `hdfCompressionLevel` respectively give the chunk size and compression level to use when writing the file.

A.11.1. File Formats

The merger tree file builder engine can currently export in one of two formats. These formats can be specified on the command line to both `Millennium_Merger_Tree_File_Maker.F90`, and `Simple_Merger_Tree_File_Maker.F90`:

`galacticus` merger trees are exported in Galacticus's native format described in detail in §A.1;

`irate` merger trees are exported in the `IRATE` format.

A.11.2. Exporting Trees from Galacticus

By setting `[mergerTreesWrite]`=`true`, Galacticus will export each merger tree generated to the file specified by `[mergerTreeExportFileName]` using the format specified by `[mergerTreeExportOutputFormat]`. Currently, node indices (plus host indices, which are assumed identical to the node indices), descendent indices, masses and redshifts are exported. Positions and velocities are exported if available. If `IRATE`-format output is requested then "snapshot" numbers will be assigned to nodes based on the time at which they exist. This usually only makes sense if the nodes are defined on a time grid (i.e. if merger trees were extracted from an N-body simulation, or if trees were re-gridded onto such a time grid; see §12.38.2). Export happens after any merger tree pre-evolution tasks (see §16.4.3).

Bibliography

Tom Abel, Peter Anninos, Yu Zhang, and Michael L. Norman. Modeling primordial gas in numerical cosmology. *New Astronomy*, 2:181–207, August 1997. URL http://adsabs.harvard.edu/abs/1997NewA....2..181A. 3, 12.39.2, 16.4.1, 18.1, *a*

S. M. Abrarov and B. M. Quine. A rapid and highly accurate approximation for the error function of complex argument. *ArXiv e-prints*, 1308:3399, August 2013. URL http://adsabs.harvard.edu/abs/2013arXiv1308.3399A. 18.1

Jennifer K. Adelman-McCarthy, Marcel A. Agüeros, Sahar S. Allam, Carlos Allende Prieto, Kurt S. J. Anderson, Scott F. Anderson, James Annis, Neta A. Bahcall, C. A. L. Bailer-Jones, Ivan K. Baldry, J. C. Barentine, Bruce A. Bassett, Andrew C. Becker, Timothy C. Beers, Eric F. Bell, Andreas A. Berlind, Mariangela Bernardi, Michael R. Blanton, John J. Bochanski, William N. Boroski, Jarle Brinchmann, J. Brinkmann, Robert J. Brunner, Tamás Budavári, Samuel Carliles, Michael A. Carr, Francisco J. Castander, David Cinabro, R. J. Cool, Kevin R. Covey, István Csabai, Carlos E. Cunha, James R. A. Davenport, Ben Dilday, Mamoru Doi, Daniel J. Eisenstein, Michael L. Evans, Xiaohui Fan, Douglas P. Finkbeiner, Scott D. Friedman, Joshua A. Frieman, Masataka Fukugita, Boris T. Gänsicke, Evalyn Gates, Bruce Gillespie, Karl Glazebrook, Jim Gray, Eva K. Grebel, James E. Gunn, Vijay K. Gurbani, Patrick B. Hall, Paul Harding, Michael Harvanek, Suzanne L. Hawley, Jeffrey Hayes, Timothy M. Heckman, John S. Hendry, Robert B. Hindsley, Christopher M. Hirata, Craig J. Hogan, David W. Hogg, Joseph B. Hyde, Shin-ichi Ichikawa, Å¡eljko IveziÄĞ, Sebastian Jester, Jennifer A. Johnson, Anders M. Jorgensen, Mario JuriÄĞ, Stephen M. Kent, R. Kessler, S. J. Kleinman, G. R. Knapp, Richard G. Kron, Jurek Krzesinski, Nikolay Kuropatkin, Donald Q. Lamb, Hubert Lampeitl, Svetlana Lebedeva, Young Sun Lee, R. French Leger, Sébastien Lépine, Marcos Lima, Huan Lin, Daniel C. Long, Craig P. Loomis, Jon Loveday, Robert H. Lupton, Olena Malanushenko, Viktor Malanushenko, Rachel Mandelbaum, Bruce Margon, John P. Marriner, David Martínez-Delgado, Takahiko Matsubara, Peregrine M. McGehee, Timothy A. McKay, Avery Meiksin, Heather L. Morrison, Jeffrey A. Munn, Reiko Nakajima, Eric H. Neilsen, Heidi Jo Newberg, Robert C. Nichol, Tom Nicinski, Maria Nieto-Santisteban, Atsuko Nitta, Sadanori Okamura, Russell Owen, Hiroaki Oyaizu, Nikhil Padmanabhan, Kaike Pan, Changbom Park, John Peoples, Jeffrey R. Pier, Adrian C. Pope, Norbert Purger, M. Jordan Raddick, Paola Re Fiorentin, Gordon T. Richards, Michael W. Richmond, Adam G. Riess, Hans-Walter Rix, Constance M. Rockosi, Masao Sako, David J. Schlegel, Donald P. Schneider, Matthias R. Schreiber, Axel D. Schwope, UroÅą Seljak, Branimir Sesar, Erin Sheldon, Kazu Shimasaku, Thirupathi Sivarani, J. Allyn Smith, Stephanie A. Snedden, Matthias Steinmetz, Michael A. Strauss, Mark SubbaRao, Yasushi Suto, Alexander S. Szalay, István Szapudi, Paula Szkody, Max Tegmark, Aniruddha R. Thakar, Christy A. Tremonti, Douglas L. Tucker, Alan Uomoto, Daniel E. Vanden Berk, Jan Vandenberg, S. Vidrih, Michael S. Vogeley, Wolfgang Voges, Nicole P. Vogt, Yogesh Wadadekar, David H. Weinberg, Andrew A. West, Simon D. M. White, Brian C. Wilhite, Brian Yanny, D. R. Yocum, Donald G. York, Idit Zehavi, and Daniel B. Zucker. The sixth data release of the sloan digital sky survey. *The Astrophysical Journal Supplement Series*, 175:297–313, April 2008. ISSN 0067-0049. doi: 10.1086/524984;. URL http://adsabs.harvard.edu/abs/2008ApJS..175..297A. 12.59.1

S. M. V. Aldrovandi and D. Pequignot. Radiative and dielectronic recombination coefficients for complex ions. *Astronomy and Astrophysics*, 25:137, May 1973. ISSN 0004-6361. URL http://adsabs.harvard.edu/abs/1973%26A....25..137A. 16.4.1, 18.1

M. Arnaud and J. Raymond. Iron ionization and recombination rates and ionization equilibrium. *The Astrophysical Journal*, 398:394–406, October 1992. URL http://adsabs.harvard.edu/abs/1992ApJ...398..394A. 18.1

M. Arnaud and R. Rothenflug. An updated evaluation of recombination and ionization rates. *Astronomy and Astrophysics Supplement Series*, 60:425–457, June 1985. ISSN 0365-0138. URL http://adsabs.harvard.edu/abs/1985%26AS...60..425A. 16.4.1, 18.1

I. K. Baldry, S. P. Driver, J. Loveday, E. N. Taylor, L. S. Kelvin, J. Liske, P. Norberg, A. S. G. Robotham, S. Brough, A. M. Hopkins, S. P. Bamford, J. A. Peacock, J. Bland-Hawthorn, C. J. Conselice, S. M. Croom, D. H. Jones, H. R. Parkinson, C. C. Popescu, M. Prescott, R. G. Sharp, and R. J. Tuffs. Galaxy and mass assembly (GAMA): the galaxy stellar mass function at z < 0.06. *Monthly Notices of the Royal Astronomical Society*, 421:621–634, March 2012. doi: DOI:10.1111/j.1365-2966.2012.20340.x;eprintid: arXiv:1111.5707. URL http://adsabs.harvard.edu/abs/2012MNRAS.421..621B. 9.5.3, 12.59.5, 12.5, 16.4.1, 16.4.1, 18.1

J. M. Bardeen, J. R. Bond, N. Kaiser, and A. S. Szalay. The statistics of peaks of gaussian random fields. *Astrophysical Journal*, 304:15–61, May 1986. URL http://adsabs.harvard.edu/abs/1986ApJ...304...15B. 12.6.5, 16.4.1, 18.1

James M. Bardeen. Kerr metric black holes. *Nature*, 226:64, April 1970. URL http://adsabs.harvard.edu/abs/1970Natur.226...64B. 12.5.1

Rennan Barkana, Zoltán Haiman, and Jeremiah P. Ostriker. Constraints on warm dark matter from cosmological reionization. *The Astrophysical Journal*, 558:482–496, September 2001. URL http://adsabs.harvard.edu/abs/2001ApJ...558..482B. 3, 12.6.5, 12.6.8, 12.6.8, 12.6.8, 16.4.1, 18.1

Joshua E. Barnes. Gravitational softening as a smoothing operation. *Monthly Notices of the Royal Astronomical Society*, 425:1104–1120, September 2012. ISSN 0035-8711. doi: 10.1111/j.1365-2966.2012.21462.x. URL http://adsabs.harvard.edu/abs/2012MNRAS.425.1104B. 3, 18.1

C. M. Baugh, C. G. Lacey, C. S. Frenk, G. L. Granato, L. Silva, A. Bressan, A. J. Benson, and S. Cole. Can the faint submillimetre galaxies be explained in the ÎŻ cold dark matter model? *Monthly Notices of the Royal Astronomical Society*, 356:1191–1200, January 2005. URL http://adsabs.harvard.edu/abs/2005MNRAS.356.1191B. 3, 12.28.2, 12.46.5, 16.4.1, 16.4.1, 16.4.1, 18.1, 18.1, 18.1, 18.1, 18.1

Mitchell C. Begelman. Accreting Black Holes. *ArXiv e-prints*, 1410:arXiv:1410.8132, October 2014. URL http://adsabs.harvard.edu/abs/2014arXiv1410.8132B. 12.5.3

Peter S. Behroozi, Charlie Conroy, and Risa H. Wechsler. A comprehensive analysis of uncertainties affecting the stellar mass-halo mass relation for 0 < z < 4. *The Astrophysical Journal*, 717:379–403, July 2010. URL http://adsabs.harvard.edu/abs/2010ApJ...717..379B. 3, 9.5.3, 9.5.3, 9.5.3, 9.5.3, 9.5.3, 9.7.1, 9.7.1, 12.8.1, 12.1, 16.4.1, 17.1.1, 17.1, 18.1

A. J. Benson. Orbital parameters of infalling dark matter substructures. *MNRAS*, 358:551–562, April 2005. URL http://adsabs.harvard.edu/abs/2005MNRAS.358..551B. 3, 12.51.2, 16.4.1, 18.1

A. J. Benson, C. G. Lacey, C. M. Baugh, S. Cole, and C. S. Frenk. The effects of photoionization on galaxy formation - i. model and results at z=0. *MNRAS*, 333:156–176, June 2002. URL http://adsabs.harvard.edu/abs/2002MNRAS.333..156B. 12.2.1

Andrew J. Benson and Arif Babul. Maximum spin of black holes driving jets. *Monthly Notices of the Royal Astronomical Society*, 397:1302–1313, August 2009. URL http://adsabs.harvard.edu/abs/2009MNRAS.397.1302B. 3, 12.5.2, 16.4.1

Andrew J. Benson and Richard Bower. Cold mode accretion in galaxy formation. http://adsabs.harvard.edu/abs/2010arXiv1004.1162B, April 2010. URL http://adsabs.harvard.edu/abs/2010arXiv1004.1162B. 3, 11.1.1, 11.1.2, 12.2.3

Andrew J. Benson, Arya Farahi, Shaun Cole, Leonidas A. Moustakas, Adrian Jenkins, Mark Lovell, Rachel Kennedy, John Helly, and Carlos Frenk. Dark matter halo merger histories beyond cold dark matter: I - methods and application to warm dark matter. *ArXiv e-prints*, 1209:3018, September 2012. URL http://adsabs.harvard.edu/abs/2012arXiv1209.3018B. 12.14.2, 16.4.1, 18.1, 18.1

M. Bernardi, A. Meert, R. K. Sheth, V. Vikram, M. Huertas-Company, S. Mei, and F. Shankar. The massive end of the luminosity and stellar mass functions: dependence on the fit to the light profile. *Monthly Notices of the Royal Astronomical Society*, 436:697–704, November 2013. ISSN 0035-8711. doi: 10.1093/mnras/stt1607. URL http://adsabs.harvard.edu/abs/2013MNRAS.436..697B. 9.5.3, 12.59.2, 12.2, 16.4.1, 16.4.1, 18.1

Philip Bett, Vincent Eke, Carlos S. Frenk, Adrian Jenkins, John Helly, and Julio Navarro. The spin and shape of dark matter haloes in the millennium simulation of a lambda cold dark matter universe. *MNRAS*, 376:215–232, March 2007. URL http://adsabs.harvard.edu/abs/2007MNRAS.376..215B. 3, 12.11.7, 16.4.1, 18.1

Suman Bhattacharya, Katrin Heitmann, Martin White, Zarija LukiÄĞ, Christian Wagner, and Salman Habib. Mass Function Predictions Beyond ÎŻCDM. *The Astrophysical Journal*, 732:122, May 2011. ISSN 0004-637X. doi: 10.1088/0004-637X/732/2/122. URL http://adsabs.harvard.edu/abs/2011ApJ...732..122B. 3, 15.8.2, 16.4.1, 18.1

F. Bigiel, A. Leroy, F. Walter, E. Brinks, W. J. G. de Blok, B. Madore, and M. D. Thornley. The star formation law in nearby galaxies on Sub-Kpc scales. *The Astronomical Journal*, 136:2846–2871, December 2008. URL http://adsabs.harvard.edu/abs/2008AJ....136.2846B. 3

James Binney and Scott Tremaine. *Galactic Dynamics: Second Edition*. Princeton University Press, Princeton, NJ USA, 2008. URL http://adsabs.harvard.edu/abs/2008gady.book.....B. 18.1

Yuval Birnboim and Avishai Dekel. Virial shocks in galactic haloes? *Monthly Notices of the Royal Astronomical Society*, 345:349–364, October 2003. ISSN 0035-8711. doi: 10.1046/j.1365-8711.2003.06955.x;. URL http://adsabs.harvard.edu/abs/2003MNRAS.345..349B. 3

Michael R. Blanton, David J. Schlegel, Michael A. Strauss, J. Brinkmann, Douglas Finkbeiner, Masataka Fukugita, James E. Gunn, David W. Hogg, Å¡eljko IveziÄĞ, G. R. Knapp, Robert H. Lupton, Jeffrey A. Munn, Donald P. Schneider, Max Tegmark, and Idit Zehavi. New york university value-added galaxy catalog: A galaxy catalog based on new public surveys. *The Astronomical Journal*, 129:2562–2578, June 2005. ISSN 0004-6256. doi: 10.1086/429803;. URL http://adsabs.harvard.edu/abs/2005AJ...129.2562B. 12.59.1

Leo Blitz and Erik Rosolowsky. The role of pressure in GMC formation II: the H2-Pressure relation. *The Astrophysical Journal*, 650:933–944, October 2006. URL http://adsabs.harvard.edu/abs/2006ApJ..650..933B. 3, 12.45.3, 16.4.1, 18.1

Paul Bode, Jeremiah P. Ostriker, and Neil Turok. Halo formation in warm dark matter models. *The Astrophysical Journal*, 556:93–107, July 2001. URL http://adsabs.harvard.edu/abs/2001ApJ...556...93B. 16.4.1, 18.1

P. Bouchet, J. Lequeux, E. Maurice, L. Prevot, and M. L. Prevot-Burnichon. The visible and infrared extinction law and the gas-to-dust ratio in the small magellanic cloud. *Astronomy and Astrophysics*, 149:330–336, August 1985. ISSN 0004-6361. URL http://adsabs.harvard.edu/abs/1985%26A...149..330B. 16.4.1, 18.1

Michael Boylan-Kolchin, Chung-Pei Ma, and Eliot Quataert. Dynamical friction and galaxy merging timescales. *MNRAS*, 383:93–101, 2008. URL http://adsabs.harvard.edu/abs/2008MNRAS.383...93B. 6.1.1, 12.51.1, 16.4.1, 18.1

Michael Boylan-Kolchin, Volker Springel, Simon D. M. White, and Adrian Jenkins. There's no place like home? Statistics of Milky Way-mass dark matter haloes. *Mon Not R Astron Soc*, 406(2):896–912, August 2010. ISSN 0035-8711. doi: 10.1111/j.1365-2966.2010.16774.x. URL https://academic.oup.com/mnras/article/406/2/896/999174. 3, 18.1

Stephen P. Brooks and Andrew Gelman. General methods for monitoring convergence of iterative simulations. *Journal of Computational and Graphical Statistics*, 7(4):434–455, 1998. ISSN 1061-8600. doi: 10.1080/10618600.1998.10474787. URL http://www.tandfonline.com/doi/abs/10.1080/10618600.1998.10474787. 9.7.2

G. Bruzual and S. Charlot. Stellar population synthesis at the resolution of 2003. *Monthly Notices of the Royal Astronomical Society*, 344:1000–1028, October 2003. URL http://adsabs.harvard.edu/abs/2003MNRAS.344.1000B. 16.4.1

Greg L. Bryan and Michael L. Norman. Statistical properties of X-Ray clusters: Analytic and numerical comparisons. *The Astrophysical Journal*, 495:80, March 1998. URL http://adsabs.harvard.edu/abs/1998ApJ...495...80B. 12.6.9, 15.8.1, 16.4.1, 18.1

J. S. Bullock, T. S. Kolatt, Y. Sigad, R. S. Somerville, A. V. Kravtsov, A. A. Klypin, J. R. Primack, and A. Dekel. Profiles of dark haloes: evolution, scatter and environment. *Monthly Notices of the Royal Astronomical Society*, 321:559–575, March 2001. URL http://adsabs.harvard.edu/abs/2001MNRAS.321..559B. 3, 12.11.1, 16.4.1, 16.4.1, 18.1, 18.1, 18.1

A. Burkert. The structure of dark matter halos in dwarf galaxies. *The Astrophysical Journal Letters*, 447:L25, July 1995. ISSN 0004-637X. doi: 10.1086/309560. URL http://adsabs.harvard.edu/abs/1995ApJ...447L..25B. 12.11.2, 16.4.1, 18.1

Daniela Calzetti, Lee Armus, Ralph C. Bohlin, Anne L. Kinney, Jan Koornneef, and Thaisa Storchi-Bergmann. The dust content and opacity of actively star-forming galaxies. *The Astrophysical Journal*, 533:682–695, April 2000. ISSN 0004-637X. doi: 10.1086/308692. URL http://adsabs.harvard.edu/abs/2000ApJ...533..682C. 16.4.1, 18.1

Manuela Campanelli, Carlos Lousto, Yosef Zlochower, and David Merritt. Large merger recoils and spin flips from generic black hole binaries. *The Astrophysical Journal*, 659:L5–L8, April 2007. URL http://adsabs.harvard.edu/abs/2007ApJ...659L...5C. 12.56.2, 18.1

Michele Cappellari, Eric Emsellem, R. Bacon, M. Bureau, Roger L. Davies, P. T. de Zeeuw, Jesús Falcón-Barroso, Davor KrajnoviÄĞ, Harald Kuntschner, Richard M. McDermid, Reynier F. Peletier, Marc Sarzi, Remco C. E. van den Bosch, and Glenn van de Ven. The SAURON project - x. the orbital anisotropy of elliptical and lenticular galaxies: revisiting the (v/ÏČ, ÉŻ) diagram with integral-field stellar kinematics. *Monthly Notices of the Royal Astronomical Society*, 379:418–444, August 2007. ISSN 0035-8711. doi: 10.1111/j.1365-2966.2007.11963.x;. URL http://adsabs.harvard.edu/abs/2007MNRAS.379..418C. 13.22, 18.1, 18.1

K. I. Caputi, M. Cirasuolo, J. S. Dunlop, R. J. McLure, D. Farrah, and O. Almaini. The stellar mass function of the most-massive galaxies at 3 âĽďz < 5 in the UKIDSS ultra deep survey. *Monthly Notices of the Royal Astronomical Society*, 413:162–176, May 2011. URL http://adsabs.harvard.edu/abs/2011MNRAS.413..162C. 9.5.3, 9.5.3, 9.5.3, 9.5.3, 9.3, 9.5.3, 9.4, 12.59.9, 12.59.9, 12.4, 12.9, 16.4.1, 18.1

Jason A. Cardelli, Geoffrey C. Clayton, and John S. Mathis. The relationship between infrared, optical, and ultraviolet extinction. *The Astrophysical Journal*, 345:245–256, October 1989. ISSN 0004-637X. doi: 10.1086/167900. URL http://adsabs.harvard.edu/abs/1989ApJ...345..245C. 15.8.2, 16.4.1, 18.1

V. F. Cardone, E. Piedipalumbo, and C. Tortora. Spherical galaxy models with power-law logarithmic slope. *Monthly Notices of the Royal Astronomical Society*, 358:1325–1336, April 2005. URL http://adsabs.harvard.edu/abs/2005MNRAS.358.1325C. 12.11.2

Gilles Chabrier. The galactic disk mass budget. i. stellar mass function and density. *The Astrophysical Journal*, 554:1274–1281, June 2001. URL http://adsabs.harvard.edu/abs/2001ApJ...554.1274C. 3, 12.28.2, 16.4.1, 18.1

Kwan Chuen Chan, Ravi K. Sheth, and Román Scoccimarro. Effective window function for Lagrangian halos. *Physical Review D*, 96:103543, November 2017. ISSN 1550-7998. doi: 10.1103/PhysRevD.96.103543. URL http://adsabs.harvard.edu/abs/2017PhRvD..96j3543C. 16.4.1, 18.1

S. Chandrasekhar. Dynamical friction. i. general considerations: the coefficient of dynamical friction. *The Astrophysical Journal*, 97:255, March 1943. URL http://adsabs.harvard.edu/abs/1943ApJ...97..255C. 3, 12.41.2, 16.4.1, 18.1

Stéphane Charlot and S. Michael Fall. A simple model for the absorption of starlight by dust in galaxies. *The Astrophysical Journal*, 539:718–731, August 2000. URL http://adsabs.harvard.edu/abs/2000ApJ...539..718C. 16.4.1, 16.4.1, 18.1, 18.1

Luca Ciotti, Jeremiah P. Ostriker, and Daniel Proga. Feedback from central black holes in elliptical galaxies. i. models with either radiative or mechanical feedback but not both. *The Astrophysical Journal*, 699:89–104, July 2009. URL http://adsabs.harvard.edu/abs/2009ApJ...699...89C. 11.1.1

Shaun Cole, Cedric G. Lacey, Carlton M. Baugh, and Carlos S. Frenk. Hierarchical galaxy formation. *MNRAS*, 319:168–204, November 2000. URL http://adsabs.harvard.edu/abs/2000MNRAS.319..168C. 3, 11.4.3, 11.4.3, 12.9.2, 12.18.2, 12.18.2, 12.18.3, 4, 12.36.1, 12.36.1, 12.51.1, 12.51.1, 16.4.1, 16.4.1, 16.4.1, 16.4.1, 16.4.1, 16.4.1, 18.1, 18.1, 18.1, 18.1, 18.1, 18.1, 18.1, 18.1, 18.1, 18.1, 18.1, 18.1

Johan Comparat, Francisco Prada, Gustavo Yepes, and Anatoly Klypin. Accurate mass and velocity functions of dark matter haloes. *Monthly Notices of the Royal Astronomical Society*, 469:4157–4174, August 2017. ISSN 0035-8711. doi: 10.1093/mnras/stx1183. URL http://adsabs.harvard.edu/abs/2017MNRAS.469.4157C. 3

Charlie Conroy, James E. Gunn, and Martin White. The propagation of uncertainties in stellar population synthesis modeling. i. the relevance of uncertain aspects of stellar evolution and the initial mass function to the derived physical properties of galaxies. *Astrophysical Journal*, 699:486–506, July 2009. URL http://adsabs.harvard.edu/abs/2009ApJ...699..486C. 12.48.1, 16.4.1, 18.1, 18.1

Asantha Cooray and Ravi Sheth. Halo models of large scale structure. *Physics Reports*, 372:1–129, December 2002. URL http://adsabs.harvard.edu/abs/2002PhR...372....1C. 9.5.3, 18.1, 18.1, 18.1, 18.1

Camila A. Correa, J. Stuart B. Wyithe, Joop Schaye, and Alan R. Duffy. The accretion history of dark matter halos III: A physical model for the concentration-mass relation. *ArXiv e-prints*, 1502:391, February 2015. URL http://adsabs.harvard.edu/abs/2015arXiv150200391C. 16.4.1, 16.4.1, 18.1, 18.1, 18.1

M. Covington, A. Dekel, T. J. Cox, P. Jonsson, and J. R. Primack. Predicting the properties of the remnants of dissipative galaxy mergers. *Monthly Notices of the Royal Astronomical Society*, 384:94–106, February 2008. URL http://adsabs.harvard.edu/abs/2008MNRAS.384...94C. 3, 12.18.2, 18.1

M. Covington, J. R. Primack, Lauren Porter, Darren Croton, R. S. Somerville, and Avishai Dekel. The effects of dissipation on Early-Type scaling relations in Semi-Analytic models. *MNRAS*, in preparation, 2011. 12.18.2, 12.18.2

Peter Creasey, Tom Theuns, and Richard G. Bower. How supernova explosions power galactic winds. *ArXiv e-prints*, 1211:1395, November 2012. URL http://adsabs.harvard.edu/abs/2012arXiv1211.1395C. 3, 12.52.3, 16.4.1, 18.1

Z. J. Czech, G. Havas, and B. S. Majewski. Fundamental study perfect hashing. *Theoretical Computer Science*, 182:1–143, 1997. 18.1

I. Davidzon, M. Bolzonella, J. Coupon, O. Ilbert, S. Arnouts, S. de la Torre, A. Fritz, G. De Lucia, A. Iovino, B. R. Granett, G. Zamorani, L. Guzzo, U. Abbas, C. Adami, J. Bel, D. Bottini, E. Branchini, A. Cappi, O. Cucciati, P. Franzetti, M. Fumana, B. Garilli, J. Krywult, V. Le Brun, O. Le Fèvre, D. Maccagni, K. MaÅĆek, F. Marulli, H. J. McCracken, L. Paioro, J. A. Peacock, M. Polletta, A. Pollo, H. Schlagenhaufer, M. Scodeggio, L. A. M. Tasca, R. Tojeiro, D. Vergani, A. Zanichelli, A. Burden, C. Di Porto, A. Marchetti, C. Marinoni, Y. Mellier, L. Moscardini, T. Moutard, R. C. Nichol, W. J. Percival, S. Phleps, and M. Wolk. The VIMOS public extragalactic redshift survey (VIPERS). a precise measurement of the galaxy stellar mass function and the abundance of massive galaxies at redshifts $0.5 < z < 1.3$. *Astronomy and Astrophysics*, 558:23, October 2013. ISSN 0004-6361. doi: 10.1051/0004-6361/201321511. URL http://adsabs.harvard.edu/abs/2013%26A...558A..23D. 9.5.3, 12.59.3, 12.3, 16.4.1, 18.1

Gabriella De Lucia and Jérémy Blaizot. The hierarchical formation of the brightest cluster galaxies. *Monthly Notices of the Royal Astronomical Society*, 375:2–14, February 2007. ISSN 0035-8711. doi: 10.1111/j.1365-2966.2006.11287.x;. URL http://adsabs.harvard.edu/abs/2007MNRAS.375....2D. 12.59.1, 12.1

Giulia Despali, Carlo Giocoli, Raul E. Angulo, Giuseppe Tormen, Ravi K. Sheth, Giacomo Baso, and Lauro Moscardini. The universality of the virial halo mass function and models for non-universality of other halo definitions. *ArXiv e-prints*, 1507:5627, July 2015. URL http://adsabs.harvard.edu/abs/2015arXiv150705627D. 12.6.11, 15.8.2, 16.4.1, 18.1

Benedikt Diemer and Andrey V. Kravtsov. A universal model for halo concentrations. *arXiv:1407.4730 [astro-ph]*, July 2014. URL http://arxiv.org/abs/1407.4730. arXiv: 1407.4730. 3, 12.11.3, 12.11.3, 16.4.1, 18.1, 18.1

S. P. Driver, D. T. Hill, L. S. Kelvin, A. S. G. Robotham, J. Liske, P. Norberg, I. K. Baldry, S. P. Bamford, A. M. Hopkins, J. Loveday, J. A. Peacock, E. Andrae, J. Bland-Hawthorn, S. Brough, M. J. I. Brown, E. Cameron, J. H. Y. Ching, M. Colless, C. J. Conselice, S. M. Croom, N. J. G. Cross, R. de Propris, S. Dye, M. J. Drinkwater, S. Ellis, Alister W. Graham, M. W. Grootes, M. Gunawardhana, D. H. Jones, E. van Kampen, C. Maraston, R. C. Nichol, H. R. Parkinson, S. Phillipps, K. Pimbblet, C. C. Popescu, M. Prescott, I. G. Roseboom, E. M. Sadler, A. E. Sansom, R. G. Sharp, D. J. B. Smith, E. Taylor, D. Thomas, R. J. Tuffs, D. Wijesinghe, L. Dunne, C. S. Frenk, M. J. Jarvis, B. F. Madore, M. J. Meyer, M. Seibert, L. Staveley-Smith, W. J. Sutherland, and S. J. Warren. Galaxy and mass assembly (GAMA): survey diagnostics and core data release. *Monthly Notices of the Royal Astronomical Society*, 413:971–995, May 2011. ISSN 0035-8711. doi: 10.1111/j.1365-2966.2010.18188.x. URL http://adsabs.harvard.edu/abs/2011MNRAS.413..971D. 12.59.5

Xiaolong Du, Christoph Behrens, and Jens C. Niemeyer. Substructure of fuzzy dark matter haloes. *Monthly Notices of the Royal Astronomical Society*, 465:941–951, February 2017. ISSN 0035-8711. doi: 10.1093/mnras/stw2724. URL http://adsabs.harvard.edu/abs/2017MNRAS.465..941D. 16.4.1, 18.1

Aaron A. Dutton and Andrea V. Macciò. Cold dark matter haloes in the planck era: evolution of structural parameters for einasto and NFW profiles. *ArXiv e-prints*, 1402:7073, February 2014. URL http://adsabs.harvard.edu/abs/2014arXiv1402.7073D. 3, 12.11.3, 12.11.3, 12.11.3, 12.3, 16.4.1, 18.1

Richard Edgar. A review of Bondi-Hoyle-Lyttleton accretion. *New Astronomy Reviews*, 48(10):843–859, September 2004. ISSN 1387-6473. doi: 10.1016/j.newar.2004.06.001. 18.1

G. Efstathiou, G. Lake, and J. Negroponte. The stability and masses of disc galaxies. *Monthly Notices of the Royal Astronomical Society*, 199:1069–1088, June 1982. URL http://adsabs.harvard.edu/abs/1982MNRAS.199.1069E. 3, 12.12.2, 12.12.3, 15.8.2, 15.8.2, 16.4.1, 18.1

Daniel J. Eisenstein and Wayne Hu. Power spectra for cold dark matter and its variants. *The Astrophysical Journal*, 511:5–15, January 1999. URL http://adsabs.harvard.edu/abs/1999ApJ...511....5E. 12.6.5, 15.8.2, 16.4.1, 18.1

M. Fernandez and S. Williams. Closed-form expression for the poisson-binomial probability density function. *IEEE Transactions on Aerospace and Electronic Systems*, 46(2):803–817, April 2010. ISSN 0018-9251. doi: 10.1109/TAES.2010.5461658. 18.1

D. J. Fixsen. The temperature of the cosmic microwave background. *The Astrophysical Journal*, 707: 916–920, December 2009. URL http://adsabs.harvard.edu/abs/2009ApJ...707..916F. 3

A. S. Font, R. G. Bower, I. G. McCarthy, A. J. Benson, C. S. Frenk, J. C. Helly, C. G. Lacey, C. M. Baugh, and S. Cole. The colours of satellite galaxies in groups and clusters. *MNRAS*, 389:1619–1629, October 2008. URL http://adsabs.harvard.edu/abs/2008MNRAS.389.1619F. 3, 11.3.3, 12.23.2, 12.24.2, 16.4.1, 16.4.1, 18.1

Andreea Font, Andrew Benson, Richard Bower, and Carlos Frenk. Modeling the milky way satellites with Self-Consistent reionization. *in prep*, 2010. 12.2.1

Daniele Galli and Francesco Palla. The chemistry of the early universe. *Astronomy and Astrophysics*, 335: 403–420, July 1998. URL http://adsabs.harvard.edu/abs/1998%26A...335..403G. 12.9.1, 16.4.1, 18.1

Liang Gao, Julio F. Navarro, Shaun Cole, Carlos S. Frenk, Simon D. M. White, Volker Springel, Adrian Jenkins, and Angelo F. Neto. The redshift dependence of the structure of massive lambda cold dark matter haloes. *MNRAS*, 387:536–544, June 2008. URL http://adsabs.harvard.edu/abs/2008MNRAS.387..536G. 12.11.3, 12.11.3, 12.11.4, 12.11.4, 16.4.1, 16.4.1, 18.1

A. Gelman and D. B. Rubin. Inference from iterative simulation using multiple sequences. *Statistical Science*, 7(4):457–472, 1992. 9.7.2

Stuart P. D. Gill, Alexander Knebe, and Brad K. Gibson. The evolution of substructure - i. a new identification method. *Monthly Notices of the Royal Astronomical Society*, 351:399–409, June 2004. ISSN 0035-8711. doi: 10.1111/j.1365-2966.2004.07786.x. URL http://adsabs.harvard.edu/abs/2004MNRAS.351..399G. A.11.2

Carlo Giocoli, Giuseppe Tormen, and Frank C. van den Bosch. The population of dark matter subhaloes: mass functions and average mass-loss rates. *Monthly Notices of the Royal Astronomical Society*, 386: 2135–2144, June 2008. ISSN 0035-8711. doi: 10.1111/j.1365-2966.2008.13182.x. URL http://adsabs.harvard.edu/abs/2008MNRAS.386.2135G. 3, 16.4.1, 18.1

Nickolay Y. Gnedin. Effect of Reionization on Structure Formation in the Universe. *The Astrophysical Journal*, 542:535–541, October 2000. ISSN 0004-637X. doi: 10.1086/317042. URL http://adsabs.harvard.edu/abs/2000ApJ...542..535G. 12.2.4, 16.4.1, 18.1

Oleg Y. Gnedin, Lars Hernquist, and Jeremiah P. Ostriker. Tidal shocking by extended mass distributions. *The Astrophysical Journal*, 514:109–118, March 1999. ISSN 0004-637X. doi: 10.1086/306910. URL http://adsabs.harvard.edu/abs/1999ApJ...514..109G. 12.44.2, 16.4.1, 18.1

Oleg Y. Gnedin, Andrey V. Kravtsov, Anatoly A. Klypin, and Daisuke Nagai. Response of dark matter halos to condensation of baryons: Cosmological simulations and improved adiabatic contraction model. *Astrophysical Journal*, 616:16–26, November 2004. URL http://adsabs.harvard.edu/abs/2004ApJ...616...16G. 3, 12.17.2, 18.1

José A. González, Mark Hannam, Ulrich Sperhake, Bernd Brügmann, and Sascha Husa. Supermassive recoil velocities for binary Black-Hole mergers with antialigned spins. *Physical Review Letters*, 98: 231101, June 2007a. URL http://adsabs.harvard.edu/abs/2007PhRvL..98w1101G. 12.56.2, 18.1

José A. González, Ulrich Sperhake, Bernd Brügmann, Mark Hannam, and Sascha Husa. Maximum kick from nonspinning Black-Hole binary inspiral. *Physical Review Letters*, 98:91101, March 2007b. URL http://adsabs.harvard.edu/abs/2007PhRvL..98i1101G. 12.56.2, 18.1

Karl D. Gordon, Geoffrey C. Clayton, K. A. Misselt, Arlo U. Landolt, and Michael J. Wolff. A quantitative comparison of the small magellanic cloud, large magellanic cloud, and milky way ultraviolet to near-infrared extinction curves. *ApJ*, 594(1):279, September 2003. ISSN 0004-637X. doi: 10.1086/376774. URL http://iopscience.iop.org/0004-637X/594/1/279. 3, 16.4.1, 18.1

M. L. P. Gunawardhana, A. M. Hopkins, J. Bland-Hawthorn, S. Brough, R. Sharp, J. Loveday, E. Taylor, D. H. Jones, M. A. Lara-López, A. E. Bauer, M. Colless, M. Owers, I. K. Baldry, A. R. López-Sánchez, C. Foster, S. Bamford, M. J. I. Brown, S. P. Driver, M. J. Drinkwater, J. Liske, M. Meyer, P. Norberg, A. S. G. Robotham, J. H. Y. Ching, M. E. Cluver, S. Croom, L. Kelvin, M. Prescott, O. Steele, D. Thomas, and L. Wang. Galaxy And Mass Assembly: evolution of the HÎś luminosity function and star formation rate density up to z $<$ 0.35. *Monthly Notices of the Royal Astronomical Society*, 433:2764–2789, August 2013. ISSN 0035-8711. doi: 10.1093/mnras/stt890. URL http://adsabs.harvard.edu/abs/2013MNRAS.433.2764G. 16.4.1, 16.4.1, 18.1

Qi Guo, Simon White, Michael Boylan-Kolchin, Gabriella De Lucia, Guinevere Kauffmann, Gerard Lemson, Cheng Li, Volker Springel, and Simone Weinmann. From dwarf spheroidals to cD galaxies: simulating the galaxy population in a ÎŻCDM cosmology. *Monthly Notices of the Royal Astronomical Society*, 413:101–131, May 2011. ISSN 0035-8711. doi: 10.1111/j.1365-2966.2010.18114.x. URL http://adsabs.harvard.edu/abs/2011MNRAS.413..101G. 9.5.3, 12.59.9, 12.9

Michael Gustafsson, Malcolm Fairbairn, and Jesper Sommer-Larsen. Baryonic pinching of galactic dark matter halos. *Physical Review D*, 74:123522, December 2006. URL http://adsabs.harvard.edu/abs/2006PhRvD..74l3522G. 3

L. Guzzo, M. Scodeggio, B. Garilli, B. R. Granett, U. Abbas, C. Adami, S. Arnouts, J. Bel, M. Bolzonella, D. Bottini, E. Branchini, A. Cappi, J. Coupon, O. Cucciati, I. Davidzon, G. De Lucia, S. de la Torre, A. Fritz, P. Franzetti, M. Fumana, P. Hudelot, O. Ilbert, A. Iovino, J. Krywult, V. Le Brun, O. Le Fèvre,

D. Maccagni, K. MaÅĆek, F. Marulli, H. J. McCracken, L. Paioro, J. A. Peacock, M. Polletta, A. Pollo, H. Schlagenhaufer, L. A. M. Tasca, R. Tojeiro, D. Vergani, G. Zamorani, A. Zanichelli, A. Burden, C. Di Porto, A. Marchetti, C. Marinoni, Y. Mellier, L. Moscardini, R. C. Nichol, W. J. Percival, S. Phleps, and M. Wolk. The VIMOS public extragalactic redshift survey (VIPERS). an unprecedented view of galaxies and large-scale structure at 0.5. *ArXiv e-prints*, 1303:2623, March 2013. URL http://adsabs.harvard.edu/abs/2013arXiv1303.2623G. 12.59.3

Martha P. Haynes, Riccardo Giovanelli, Ann M. Martin, Kelley M. Hess, Amélie Saintonge, Elizabeth A. K. Adams, Gregory Hallenbeck, G. Lyle Hoffman, Shan Huang, Brian R. Kent, Rebecca A. Koopmann, Emmanouil Papastergis, Sabrina Stierwalt, Thomas J. Balonek, David W. Craig, Sarah J. U. Higdon, David A. Kornreich, Jeffrey R. Miller, Aileen A. O'Donoghue, Ronald P. Olowin, Jessica L. Rosenberg, Kristine Spekkens, Parker Troischt, and Eric M. Wilcots. The arecibo legacy fast ALFA survey: The Îś.40 h i source catalog, its characteristics and their impact on the derivation of the h i mass function. *The Astronomical Journal*, 142:170, November 2011. ISSN 0004-6256. doi: 10.1088/0004-6256/142/5/170;. URL http://adsabs.harvard.edu/abs/2011AJ....142..170H. 4.2, 4.4.1, 12.59.8

Andrew P. Hearin, Douglas F. Watson, Matthew R. Becker, Reinabelle Reyes, Andreas A. Berlind, and Andrew R. Zentner. The dark side of galaxy color: evidence from new SDSS measurements of galaxy clustering and lensing. *ArXiv e-prints*, 1310:6747, October 2013. URL http://adsabs.harvard.edu/abs/2013arXiv1310.6747H. 16.4.1, 16.4.1, 18.1

A. Heger and S. E. Woosley. The nucleosynthetic signature of population III. *Astrophysical Journal*, 567:532–543, March 2002. URL http://adsabs.harvard.edu/abs/2002ApJ...567..532H. 12.50.5, 16.4.1, 18.1

Bruno M. B. Henriques, Simon D. M. White, Gerard Lemson, Peter A. Thomas, Qi Guo, Gabriel-Dominique Marleau, and Roderik A. Overzier. Confronting theoretical models with the observed evolution of the galaxy population out to z= 4. *Monthly Notices of the Royal Astronomical Society*, 421:2904–2916, April 2012. ISSN 0035-8711. doi: 10.1111/j.1365-2966.2012.20521.x. URL http://adsabs.harvard.edu/abs/2012MNRAS.421.2904H. 9.5.3, 9.3, 12.59.9, 12.9

Bruno M. B. Henriques, Simon D. M. White, Peter A. Thomas, Raul E. Angulo, Qi Guo, Gerard Lemson, and Volker Springel. Simulations of the galaxy population constrained by observations from z = 3 to the present day: implications for galactic winds and the fate of their ejecta. *Monthly Notices of the Royal Astronomical Society*, 431:3373–3395, June 2013. ISSN 0035-8711. doi: 10.1093/mnras/stt415. URL http://adsabs.harvard.edu/abs/2013MNRAS.431.3373H. 3, 16.4.1, 18.1

Lars Hernquist. An analytical model for spherical galaxies and bulges. *Astrophysical Journal*, 356: 359–364, June 1990. URL http://adsabs.harvard.edu/abs/1990ApJ...356..359H. 11.5.2, 16.4.1, 18.1

Stefan Hilbert, Simon D. M. White, Jan Hartlap, and Peter Schneider. Strong-lensing optical depths in a LambdaCDM universe - II. the influence of the stellar mass in galaxies. *Monthly Notices of the Royal Astronomical Society*, 386:1845–1854, June 2008. ISSN 0035-8711. doi: 10.1111/j.1365-2966.2008.13190.x. URL http://adsabs.harvard.edu/abs/2008MNRAS.386.1845H. 12.19.2

G. Hinshaw, D. Larson, E. Komatsu, D. N. Spergel, C. L. Bennett, J. Dunkley, M. R. Nolta, M. Halpern, R. S. Hill, N. Odegard, L. Page, K. M. Smith, J. L. Weiland, B. Gold, N. Jarosik, A. Kogut, M. Limon, S. S. Meyer, G. S. Tucker, E. Wollack, and E. L. Wright. Nine-year wilkinson microwave anisotropy probe (WMAP) observations: Cosmological parameter results. *ArXiv e-prints*, 1212:5226, December 2012. URL http://adsabs.harvard.edu/abs/2012arXiv1212.5226H. 2.2.2, 3

Craig J. Hogan. Warm Dark Matter: Clues to Primordial Phase Density from the Structure of Galaxy Halos. *ArXiv Astrophysics e-prints*, pages arXiv:astro–ph/9912549, December 1999. URL http://adsabs.harvard.edu/abs/1999astro.ph.12549H. 18.1

David W Hogg, Ivan K Baldry, Michael R Blanton, and Daniel J Eisenstein. The k correction. *astro-ph/0210394*, October 2002. URL http://arxiv.org/abs/astro-ph/0210394. 5.1, 18.1

D. Hollenbach and C. F. McKee. Molecule formation and infrared emission in fast interstellar shocks. i physical processes. *The Astrophysical Journal Supplement Series*, 41:555–592, November 1979. URL http://adsabs.harvard.edu/abs/1979ApJS...41..555H. 18.1

Philip F. Hopkins, Gordon T. Richards, and Lars Hernquist. An observational determination of the bolometric quasar luminosity function. *The Astrophysical Journal*, 654:731–753, January 2007. URL http://adsabs.harvard.edu/abs/2007ApJ...654..731H. 16.4.1, 18.1

Cathy Horellou and Joel Berge. Dark energy and the evolution of spherical overdensities. *Monthly Notices of the Royal Astronomical Society*, 360:1393–1400, July 2005. ISSN 0035-8711. doi: 10.1111/j.1365-2966.2005.09140.x;. URL http://adsabs.harvard.edu/abs/2005MNRAS.360.1393H. 18.1

Akio K. Inoue, Ikkoh Shimizu, and Ikuru Iwata. An updated analytic model for the attenuation by the intergalactic medium. *ArXiv e-prints*, 1402:677, February 2014. URL http://adsabs.harvard.edu/abs/2014arXiv1402.0677I. 12.49.1, 16.4.1, 18.1, 18.1

H. K. Jassal, J. S. Bagla, and T. Padmanabhan. WMAP constraints on low redshift evolution of dark energy. *Monthly Notices of the Royal Astronomical Society*, 356:L11–L16, January 2005. ISSN 0035-8711. doi: 10.1111/j.1745-3933.2005.08577.x;. URL http://adsabs.harvard.edu/abs/2005MNRAS.356L..11J. 12.4.2

C. Y. Jiang, Y. P. Jing, A. Faltenbacher, W. P. Lin, and Cheng Li. A fitting formula for the merger timescale of galaxies in hierarchical clustering. *ApJ*, 675:1095–1105, March 2008. URL http://adsabs.harvard.edu/abs/2008ApJ...675.1095J. 9.3.3, 12.51.1, 16.4.1, 18.1

Fangzhou Jiang and Frank C. van den Bosch. Generating merger trees for dark matter haloes: a comparison of methods. *Monthly Notices of the Royal Astronomical Society*, 440:193–207, May 2014. ISSN 0035-8711. doi: 10.1093/mnras/stu280. URL http://adsabs.harvard.edu/abs/2014MNRAS.440..193J. 16.4.1

Lilian Jiang, Shaun Cole, Till Sawala, and Carlos S. Frenk. Orbital parameters of infalling satellite haloes in the hierarchical λCDM model. *arXiv:1409.1179 [astro-ph]*, September 2014. URL http://arxiv.org/abs/1409.1179. arXiv: 1409.1179. 3, 12.51.2, 16.4.1, 18.1

Stelios Kazantzidis, Andrew R. Zentner, and Andrey V. Kravtsov. The robustness of dark matter density profiles in dissipationless mergers. *The Astrophysical Journal*, 641(2):647, 2006. doi: 10.1086/500579. URL http://stacks.iop.org/0004-637X/641/i=2/a=647. 16.4.1, 18.1

Lee S. Kelvin, Simon P. Driver, Aaron S. G. Robotham, Edward N. Taylor, Alister W. Graham, Mehmet Alpaslan, Ivan Baldry, Steven P. Bamford, Amanda E. Bauer, Joss Bland-Hawthorn, Michael J. I. Brown, Matthew Colless, Christopher J. Conselice, Benne W. Holwerda, Andrew M. Hopkins, Maritza A. Lara-López, Jochen Liske, Ángel R. López-Sánchez, Jon Loveday, Peder Norberg, Steven Phillipps, Cristina C. Popescu, Matthew Prescott, Anne E. Sansom, and Richard J. Tuffs. Galaxy And Mass Assembly (GAMA): stellar mass functions by Hubble type. *Monthly Notices of the Royal Astronomical Society*, 444:1647–1659, October 2014. ISSN 0035-8711. doi: 10.1093/mnras/stu1507. URL http://adsabs.harvard.edu/abs/2014MNRAS.444.1647K. 16.4.1, 18.1

R. C. Kennicutt. The rate of star formation in normal disk galaxies. *The Astrophysical Journal*, 272:54–67, September 1983. URL http://adsabs.harvard.edu/abs/1983ApJ...272...54K. 12.28.2, 16.4.1, 18.1

Robert C. Kennicutt. The star formation law in galactic disks. *The Astrophysical Journal*, 344:685–703, September 1989. URL http://adsabs.harvard.edu/abs/1989ApJ...344..685K. 3

Robert C. Kennicutt. The global schmidt law in star-forming galaxies. *The Astrophysical Journal*, 498: 541, May 1998. URL http://adsabs.harvard.edu/abs/1998ApJ...498..541K. 12.45.1

Ivan King. The structure of star clusters. i. an empirical density law. *The Astronomical Journal*, 67: 471, October 1962. ISSN 0004-6256. doi: 10.1086/108756. URL http://adsabs.harvard.edu/abs/1962AJ.....67..471K. 12.43.2

Tetsu Kitayama and Yasushi Suto. Semianalytic predictions for statistical properties of x-ray clusters of galaxies in cold dark matter universes. *The Astrophysical Journal*, 469:480, October 1996. doi: DOI: 10.1086/177797;eprintid:arXiv:astro-ph/9604141. URL http://adsabs.harvard.edu/abs/1996ApJ...469..480K. 12.6.7, 12.6.9, 16.4.1, 16.4.1, 18.1, 18.1

O Klein and Y Nishina. über die streuung von strahlung durch freie elektronen nach der neuen relativistischen quantendynamik von dirac. *Z. Phys.*, 52(11-12):853–868, 1929. doi: 10.1007/BF01366453. 18.1

Anatoly Klypin, Gustavo Yepes, Stefan Gottlober, Francisco Prada, and Steffen Hess. MultiDark simulations: the story of dark matter halo concentrations and density profiles. *ArXiv e-prints*, 1411:4001, November 2014. URL http://adsabs.harvard.edu/abs/2014arXiv1411.4001K. 16.4.1, 16.4.1, 18.1, 18.1

Steffen R. Knollmann and Alexander Knebe. AHF: amiga's halo finder. *The Astrophysical Journal Supplement Series*, 182:608–624, June 2009. ISSN 0067-0049. doi: 10.1088/0067-0049/182/2/608. URL http://adsabs.harvard.edu/abs/2009ApJS..182..608K. A.11.2

E. Komatsu, K. M. Smith, J. Dunkley, C. L. Bennett, B. Gold, G. Hinshaw, N. Jarosik, D. Larson, M. R. Nolta, L. Page, D. N. Spergel, M. Halpern, R. S. Hill, A. Kogut, M. Limon, S. S. Meyer, N. Odegard, G. S. Tucker, J. L. Weiland, E. Wollack, and E. L. Wright. Seven-Year wilkinson microwave anisotropy probe (WMAP) observations: Cosmological interpretation. http://adsabs.harvard.edu/abs/2010arXiv1001.4538K, January 2010. URL http://adsabs.harvard.edu/abs/2010arXiv1001.4538K. 9.7.10

Michael Koppitz, Denis Pollney, Christian Reisswig, Luciano Rezzolla, Jonathan Thornburg, Peter Diener, and Erik Schnetter. Recoil velocities from Equal-Mass Binary-Black-Hole mergers. *Physical Review Letters*, 99:41102, July 2007. URL http://adsabs.harvard.edu/abs/2007PhRvL..99d1102K. 12.56.2, 18.1

Michiel Kregel, Pieter C. van der Kruit, and Richard de Grijs. Flattening and truncation of stellar discs in edge-on spiral galaxies. *Monthly Notices of the Royal Astronomical Society*, 334:646–668, August 2002. URL http://adsabs.harvard.edu/abs/2002MNRAS.334..646K. 3

Pavel Kroupa. On the variation of the initial mass function. *Monthly Notices of the Royal Astronomical Society*, 322:231–246, April 2001. URL http://adsabs.harvard.edu/abs/2001MNRAS.322..231K. 12.28.2, 16.4.1, 18.1

Mark R. Krumholz, Christopher F. McKee, and Jason Tumlinson. The star formation law in atomic and molecular gas. *The Astrophysical Journal*, 699:850–856, July 2009. URL http://adsabs.harvard.edu/abs/2009ApJ...699..850K. 3, 12.45.4, 16.4.1, 18.1

C. Lacey and S. Cole. Merger rates in hierarchical models of galaxy formation - part two - comparison with n-body simulations. *Monthly Notices of the Royal Astronomical Society*, 271:676, December 1994. ISSN 0035-8711. URL http://adsabs.harvard.edu/abs/1994MNRAS.271..676L. 12.6.9

Cedric Lacey and Shaun Cole. Merger rates in hierarchical models of galaxy formation. *Monthly Notices of the Royal Astronomical Society*, 262:627–649, June 1993. URL http://adsabs.harvard.edu/abs/1993MNRAS.262..627L. 12.51.1, 12.51.1, 16.4.1, 18.1

M. Landini and B. C. M. Fossi. Ion equilibrium for minor components in a thin plasma. *Astronomy and Astrophysics Supplement Series*, 91:183–196, November 1991. URL http://adsabs.harvard.edu/abs/1991%26AS...91..183L. 18.1

M. Landini and B. C. Monsignori Fossi. The X-UV spectrum of thin plasmas. *Astronomy and Astrophysics Supplement Series*, 82:229–260, February 1990. URL http://adsabs.harvard.edu/abs/1990%26AS..82..229L. 18.1

Earl Lawrence, Katrin Heitmann, Martin White, David Higdon, Christian Wagner, Salman Habib, and Brian Williams. The coyote universe. III. simulation suite and precision emulator for the nonlinear matter power spectrum. *The Astrophysical Journal*, 713:1322–1331, April 2010. doi: DOI:10.1088/0004-637X/713/2/1322;eprintid:arXiv:0912.4490. URL http://adsabs.harvard.edu/abs/2010ApJ...713.1322L. 9.5.3, 12.6.4, 16.4.1, 18.1

Alexie Leauthaud, Jeremy Tinker, Kevin Bundy, Peter S. Behroozi, Richard Massey, Jason Rhodes, Matthew R. George, Jean-Paul Kneib, Andrew Benson, Risa H. Wechsler, Michael T. Busha, Peter Capak, Marina Cortes, Olivier Ilbert, Anton M. Koekemoer, Oliver Le Fevre, Simon Lilly, Henry J. McCracken, Mara Salvato, Tim Schrabback, Nick Scoville, Tristan Smith, and James E. Taylor. New constraints on the evolution of the stellar-to-dark matter connection: a combined analysis of galaxy-galaxy lensing, clustering, and stellar mass functions from z=0.2 to z=1, April 2011. URL http://adsabs.harvard.edu/abs/2011arXiv1104.0928L. 3, 9.4.1, 9.4.1, 9.7.1, 9.7.1, 12.8.1, 17.1.1

Claus Leitherer, Carmelle Robert, and Laurent Drissen. Deposition of mass, momentum, and energy by massive stars into the interstellar medium. *ApJ*, 401:596–617, December 1992. URL http://adsabs.harvard.edu/abs/1992ApJ...401..596L. 12.50.2, 16.4.1, 18.1

Matteo Leo, Carlton M. Baugh, Baojiu Li, and Silvia Pascoli. A new smooth-k space filter approach to calculate halo abundances. *Journal of Cosmology and Astro-Particle Physics*, 04:010, April 2018. ISSN 1475-7516. doi: 10.1088/1475-7516/2018/04/010. URL http://adsabs.harvard.edu/abs/2018JCAP...04..010L. 18.1

Adam K. Leroy, Fabian Walter, Elias Brinks, Frank Bigiel, W. J. G. de Blok, Barry Madore, and M. D. Thornley. The star formation efficiency in nearby galaxies: Measuring where gas forms stars effectively. *The Astronomical Journal*, 136:2782–2845, December 2008. URL http://adsabs.harvard.edu/abs/2008AJ....136.2782L. 3

Cheng Li and Simon D. M. White. The distribution of stellar mass in the low-redshift universe. *Monthly Notices of the Royal Astronomical Society*, 398:2177–2187, October 2009. URL http://adsabs.harvard.edu/abs/2009MNRAS.398.2177L. 9.4.1, 9.4.1, 9.1, 9.5.3, 9.5.3, 12.59.1, 12.1, 16.4.1, 18.1

Ewa L. Łokas and Gary A. Mamon. Properties of spherical galaxies and clusters with an NFW density profile. *Monthly Notices of the Royal Astronomical Society*, 321(1):155–166, Feb 2001. doi: 10.1046/j.1365-8711.2001.04007.x. 18.1

Yu Lu, Andrew Benson, Yao-Yuan Mao, Stephanie Tonnesen, Annika H. G. Peter, Andrew R. Wetzel, Michael Boylan-Kolchin, and Risa H. Wechsler. The Connection between the Host Halo and the Satellite Galaxies of the Milky Way. *The Astrophysical Journal*, 830(2):59, October 2016. doi: 10.3847/0004-637X/830/2/59. URL https://ui.adsabs.harvard.edu/abs/2016ApJ...830...59L/abstract. 18.1

Aaron D. Ludlow, Julio F. Navarro, Raúl E. Angulo, Michael Boylan-Kolchin, Volker Springel, Carlos Frenk, and Simon D. M. White. The mass-concentration-redshift relation of cold dark matter haloes. *Monthly Notices of the Royal Astronomical Society*, 441:378–388, June 2014. ISSN 0035-8711. doi: 10.1093/mnras/stu483. URL http://adsabs.harvard.edu/abs/2014MNRAS.441..378L. 16.4.1, 18.1

Aaron D. Ludlow, Sownak Bose, Raúl E. Angulo, Lan Wang, Wojciech A. Hellwing, Julio F. Navarro, Shaun Cole, and Carlos S. Frenk. The mass-concentration-redshift relation of cold and warm dark matter haloes. *Monthly Notices of the Royal Astronomical Society*, 460:1214–1232, August 2016. ISSN 0035-8711. doi: 10.1093/mnras/stw1046. URL http://adsabs.harvard.edu/abs/2016MNRAS.460.1214L. 16.4.1, 16.4.1, 16.4.1, 18.1, 18.1

Piero Madau. Radiative transfer in a clumpy universe: The colors of high-redshift galaxies. *The Astrophysical Journal*, 441:18–27, March 1995. URL http://adsabs.harvard.edu/abs/1995ApJ...441...18M. 12.49.3, 16.4.1, 18.1

Gianpiero Mangano, Gennaro Miele, Sergio Pastor, Teguayco Pinto, Ofelia Pisanti, and Pasquale D. Serpico. Relic neutrino decoupling including flavour oscillations. *Nuclear Physics B*, 729(1–2):221–234, November 2005. ISSN 0550-3213. doi: 10.1016/j.nuclphysb.2005.09.041. URL http://www.sciencedirect.com/science/article/pii/S0550321305008291. 3

Claudia Maraston. Evolutionary population synthesis: models, analysis of the ingredients and application to high-z galaxies. *Monthly Notices of the Royal Astronomical Society*, 362:799–825, September 2005. URL http://adsabs.harvard.edu/abs/2005MNRAS.362..799M. 16.4.1

Ann M. Martin, Emmanouil Papastergis, Riccardo Giovanelli, Martha P. Haynes, Christopher M. Springob, and Sabrina Stierwalt. The arecibo legacy fast ALFA survey. x. the h i mass function and Ïľ_H i from the 40% ALFALFA survey. *The Astrophysical Journal*, 723:1359–1374, November 2010. ISSN 0004-637X. doi: 10.1088/0004-637X/723/2/1359;. URL http://adsabs.harvard.edu/abs/2010ApJ...723.1359M. 4.4.1, 9.5.3, 9.5.3, 9.5, 9.6, 12.59.8, 12.59.8, 12.8, 16.4.1, 18.1

A. Mazure and H. V. Capelato. Exact solutions for the spatial de vaucouleurs and sérsic laws and related quantities. *Astronomy and Astrophysics*, 383:384–389, January 2002. URL http://adsabs.harvard.edu/abs/2002%26A...383..384M. 11.5.2

Christopher F. McKee and Mark R. Krumholz. The Atomic-to-Molecular transition in galaxies. III. a new method for determining the molecular content of primordial and dusty clouds. *The Astrophysical Journal*, 709:308–320, January 2010. URL http://adsabs.harvard.edu/abs/2010ApJ...709..308M. 18.1

D. L. Meier. The association of jet production with geometrically thick accretion flows and black hole rotation. *The Astrophysical Journal*, 548:L9–L12, February 2001. URL http://adsabs.harvard.edu/abs/2001ApJ...548L...9M. 12.5.1, 18.1

Avery Meiksin. Colour corrections for high-redshift objects due to intergalactic attenuation. *Monthly Notices of the Royal Astronomical Society*, 365:807–812, January 2006. URL http://adsabs.harvard.edu/abs/2006MNRAS.365..807M. 12.49.2, 16.4.1, 18.1

G. E. Miller and J. M. Scalo. The initial mass function and stellar birthrate in the solar neighborhood. *The Astrophysical Journal Supplement Series*, 41:513–547, November 1979. URL http://adsabs.harvard.edu/abs/1979ApJS...41..513M. 12.28.2, 16.4.1, 18.1

M. Miyamoto and R. Nagai. Three-dimensional models for the distribution of mass in galaxies. *Publications of the Astronomical Society of Japan*, 27:533–543, 1975. ISSN 0004-6264. URL http://adsabs.harvard.edu/abs/1975PASJ...27..533M. 16.4.1, 18.1

H. J. Mo and S. D. M. White. An analytic model for the spatial clustering of dark matter haloes. *Monthly Notices of the Royal Astronomical Society*, 282:347–361, September 1996. URL http://adsabs.harvard.edu/abs/1996MNRAS.282..347M. 12.6.10, 16.4.1, 18.1

H. J. Mo, Shude Mao, and Simon D. M. White. The formation of galactic discs. *Monthly Notices of the Royal Astronomical Society*, 295:319–336, April 1998. doi: DOI:10.1046/j.1365-8711.1998.01227.x;eprintid:arXiv:astro-ph/9707093. URL http://adsabs.harvard.edu/abs/1998MNRAS.295..319M. 18.1

Amanda J. Moffett, Rebecca Lange, Simon P. Driver, Aaron S. G. Robotham, Lee S. Kelvin, Mehmet Alpaslan, Stephen K. Andrews, Joss Bland-Hawthorn, Sarah Brough, Michelle E. Cluver, Matthew Colless, Luke J. M. Davies, Benne W. Holwerda, Andrew M. Hopkins, Prajwal R. Kafle, Jochen Liske, and Martin Meyer. Galaxy and Mass Assembly (GAMA): the stellar mass budget of galaxy spheroids and discs. *Monthly Notices of the Royal Astronomical Society*, 462:4336–4348, November 2016. ISSN 0035-8711. doi: 10.1093/mnras/stw1861. URL http://adsabs.harvard.edu/abs/2016MNRAS.462.4336M. 16.4.1

Antonio D. Montero-Dorta and Francisco Prada. The SDSS DR6 luminosity functions of galaxies. *Monthly Notices of the Royal Astronomical Society*, 399:1106–1118, November 2009. ISSN 0035-8711. doi: 10.1111/j.1365-2966.2009.15197.x. URL http://adsabs.harvard.edu/abs/2009MNRAS.399.1106M. 16.4.1, 16.4.1, 18.1

Surhud More, Andrey V. Kravtsov, Neal Dalal, and Stefan Gottlöber. The overdensity and masses of the friends-of-friends halos and universality of halo mass function. *The Astrophysical Journal Supplement Series*, 195:4, July 2011. ISSN 0067-0049. doi: 10.1088/0067-0049/195/1/4. URL http://adsabs.harvard.edu/abs/2011ApJS..195....4M. 3, 12.6.9, 16.4.1, 16.4.1, 18.1

John Moustakas, Alison L. Coil, James Aird, Michael R. Blanton, Richard J. Cool, Daniel J. Eisenstein, Alexander J. Mendez, Kenneth C. Wong, Guangtun Zhu, and Stéphane Arnouts. PRIMUS: constraints on star formation quenching and galaxy merging, and the evolution of the stellar mass function from z = 0-1. *The Astrophysical Journal*, 767:50, April 2013. ISSN 0004-637X. doi: 10.1088/0004-637X/767/1/50. URL http://adsabs.harvard.edu/abs/2013ApJ...767...50M. 9.5.3, 9.5.3, 12.59.6, 12.6, 16.4.1, 18.1

J. C. Muñoz-Cuartas, Andrea Macci'o, Stefan Gottlöber, and Aaron Dutton. The redshift evolution of LCDM halo parameters. http://adsabs.harvard.edu/abs/2011arXiv1102.2186M, February 2011. URL http://adsabs.harvard.edu/abs/2011arXiv1102.2186M. 12.11.3, 16.4.1, 18.1

Adam Muzzin, Danilo Marchesini, Mauro Stefanon, Marijn Franx, Henry J. McCracken, Bo Milvang-Jensen, James S. Dunlop, J. P. U. Fynbo, Gabriel Brammer, Ivo Labbé, and Pieter G. van Dokkum. The evolution of the stellar mass functions of star-forming and quiescent galaxies to z = 4 from the COSMOS/UltraVISTA survey. *The Astrophysical Journal*, 777:18, November 2013. ISSN 0004-637X. doi: 10.1088/0004-637X/777/1/18. URL http://adsabs.harvard.edu/abs/2013ApJ...777...18M. 9.5.3, 9.5.3, 12.59.7, 12.7, 16.4.1, 18.1

Masahiro Nagashima, Cedric G. Lacey, Carlton M. Baugh, Carlos S. Frenk, and Shaun Cole. The metal enrichment of the intracluster medium in hierarchical galaxy formation models. *Monthly Notices of the Royal Astronomical Society*, 358:1247–1266, April 2005. URL http://adsabs.harvard.edu/abs/2005MNRAS.358.1247N. 12.50.4, 16.4.1, 18.1

S. Naoz and R. Barkana. The formation and gas content of high-redshift galaxies and minihaloes. *Monthly Notices of the Royal Astronomical Society*, 377:667–676, May 2007. ISSN 0035-8711. doi: 10.1111/j.1365-2966.2007.11636.x. URL http://adsabs.harvard.edu/abs/2007MNRAS.377..667N. 6.5.1, 12.2.4, 12.2.4, 16.4.1, 18.1, 18.1, 18.1

R. Narayan, R. Mahadevan, and E. Quataert. Advection-dominated accretion around black holess. In *Theory of Black Hole Accretion Disks*, page 148, 1998. URL http://adsabs.harvard.edu/abs/1998tbha.conf..148N. A.11.2

Ramesh Narayan and Insu Yi. Advection-dominated accretion: A self-similar solution. *Astrophysical Journal*, 428:L13–L16, June 1994. URL http://adsabs.harvard.edu/abs/1994ApJ...428L..13N. 12.5.2

Marco A. C. Nascimento and William A. Goddard. The photodetachment cross section of the negative hydrogen ion. *Physical Review A*, 16:1559–1567, October 1977. URL http://adsabs.harvard.edu/abs/1977PhRvA..16.1559N. 18.1

Julio F. Navarro, Carlos S. Frenk, and Simon D. M. White. The structure of cold dark matter halos. *The Astrophysical Journal*, 462:563, May 1996. URL http://adsabs.harvard.edu/abs/1996ApJ...462..563N. 12.11.3, 12.11.3, 16.4.1, 16.4.1, 18.1, 18.1, 18.1

Julio F. Navarro, Carlos S. Frenk, and Simon D. M. White. A universal density profile from hierarchical clustering. *ApJ*, 490:493, December 1997. URL http://adsabs.harvard.edu/abs/1997ApJ...490..493N. 12.11.2, 16.4.1, 18.1

K. Nomoto, K. Iwamoto, N. Nakasato, F.-K. Thielemann, F. Brachwitz, T. Tsujimoto, Y. Kubo, and N. Kishimoto. Nucleosynthesis in type ia supernovae. *Nuclear Physics A*, 621:467–476, February 1997. URL http://adsabs.harvard.edu/abs/1997NuPhA.621..467N. 18.1

D. Obreschkow, D. Croton, G. De Lucia, S. Khochfar, and S. Rawlings. Simulation of the cosmic evolution of atomic and molecular hydrogen in galaxies. *The Astrophysical Journal*, 698:1467–1484, June 2009. ISSN 0004-637X. doi: 10.1088/0004-637X/698/2/1467. URL http://adsabs.harvard.edu/abs/2009ApJ...698.1467O. 3, 4.4.1, 4.4.1, 18.1

Jeremiah P. Ostriker, Ena Choi, Luca Ciotti, Gregory S. Novak, and Daniel Proga. Momentum driving: which physical processes dominate AGN feedback? http://adsabs.harvard.edu/abs/2010arXiv1004.2923O, April 2010. URL http://adsabs.harvard.edu/abs/2010arXiv1004.2923O. 11.1.1, 11.1.2

Nikhil Padmanabhan, David J. Schlegel, Douglas P. Finkbeiner, J. C. Barentine, Michael R. Blanton, Howard J. Brewington, James E. Gunn, Michael Harvanek, David W. Hogg, Å¡eljko IveziÄĞ, David Johnston, Stephen M. Kent, S. J. Kleinman, Gillian R. Knapp, Jurek Krzesinski, Dan Long, Eric H. Neilsen, Atsuko Nitta, Craig Loomis, Robert H. Lupton, Sam Roweis, Stephanie A. Snedden, Michael A. Strauss, and Douglas L. Tucker. An improved photometric calibration of the sloan digital sky survey imaging data. *The Astrophysical Journal*, 674:1217–1233, February 2008. ISSN 0004-637X. doi: 10.1086/524677;. URL http://adsabs.harvard.edu/abs/2008ApJ...674.1217P. 12.59.1

Hannah Parkinson, Shaun Cole, and John Helly. Generating dark matter halo merger trees. *Monthly Notices of the Royal Astronomical Society*, 383:557–564, January 2008. URL http://adsabs.harvard.edu/abs/2008MNRAS.383..557P. 3, 12.34.1, 12.35.2, 7, 16.4.1, 16.4.1, 18.1, 18.1

Anna Patej and Abraham Loeb. A Simple Physical Model for the Gas Distribution in Galaxy Clusters. *The Astrophysical Journal Letters*, 798:L20, January 2015. ISSN 0004-637X. doi: 10.1088/2041-8205/798/1/L20. URL http://adsabs.harvard.edu/abs/2015ApJ...798L..20P. 3, 16.4.1, 18.1

J. A. Peacock and S. J. Dodds. Non-linear evolution of cosmological power spectra. *Monthly Notices of the Royal Astronomical Society*, 280:L19–L26, June 1996. doi: eprintid:arXiv:astro-ph/9603031. URL http://adsabs.harvard.edu/abs/1996MNRAS.280L..19P. 9.5.3, 12.6.4, 16.4.1, 18.1

D. Pequignot, P. Petitjean, and C. Boisson. Total and effective radiative recombination coefficients. *Astronomy and Astrophysics*, 251:680–688, November 1991. URL http://adsabs.harvard.edu/abs/1991%26A...251..680P. 18.1

W. J. Percival. Cosmological structure formation in a homogeneous dark energy background. *Astronomy and Astrophysics*, 443:819–830, December 2005. URL http://adsabs.harvard.edu/abs/2005%26A...443..819P. 3, 12.6.7, 12.6.9, 16.4.1

Will J. Percival, Robert C. Nichol, Daniel J. Eisenstein, Joshua A. Frieman, Masataka Fukugita, Jon Loveday, Adrian C. Pope, Donald P. Schneider, Alex S. Szalay, Max Tegmark, Michael S. Vogeley, David H. Weinberg, Idit Zehavi, Neta A. Bahcall, Jon Brinkmann, Andrew J. Connolly, and Avery Meiksin. The shape of the sloan digital sky survey data release 5 galaxy power spectrum. *Astrophysical Journal*, 657:645–663, March 2007. URL http://adsabs.harvard.edu/abs/2007ApJ...657..645P. 12.59.1

Warren F. Perger, Atul Bhalla, and Mark Nardin. A numerical evaluator for the generalized hypergeometric series. *Computer Physics Communications*, 77(2):249–254, October 1993. ISSN 0010-4655. doi: 10.1016/0010-4655(93)90008-Z. URL http://www.sciencedirect.com/science/article/pii/001046559390008Z. 18.1

P. C. Peters. Gravitational radiation and the motion of two point masses. *Physical Review*, 136:1224–1232, November 1964. URL http://adsabs.harvard.edu/abs/1964PhRv..136.1224P. 12.55.2

Planck Collaboration, Y. Akrami, F. Arroja, M. Ashdown, J. Aumont, C. Baccigalupi, M. Ballardini, A. J. Banday, R. B. Barreiro, N. Bartolo, S. Basak, K. Benabed, J.-P. Bernard, M. Bersanelli, P. Bielewicz, J. J. Bock, J. R. Bond, J. Borrill, F. R. Bouchet, F. Boulanger, M. Bucher, C. Burigana, R. C. Butler, E. Calabrese, J.-F. Cardoso, J. Carron, A. Challinor, H. C. Chiang, L. P. L. Colombo, C. Combet, D. Contreras, B. P. Crill, F. Cuttaia, P. de Bernardis, G. de Zotti, J. Delabrouille, J.-M. Delouis, E. Di Valentino, J. M. Diego, S. Donzelli, O. Doré, M. Douspis, A. Ducout, X. Dupac, S. Dusini, G. Efstathiou, F. Elsner, T. A. EnÃ§lin, H. K. Eriksen, Y. Fantaye, J. Fergusson, R. Fernandez-Cobos, F. Finelli, F. Forastieri, M. Frailis, E. Franceschi, A. Frolov, S. Galeotta, S. Galli, K. Ganga, C. Gauthier, R. T. Génova-Santos, M. Gerbino, T. Ghosh, J. González-Nuevo, K. M. Górski, S. Gratton, A. Gruppuso, J. E. Gudmundsson, J. Hamann, W. Handley, F. K. Hansen, D. Herranz, E. Hivon, D. C. Hooper, Z. Huang, A. H. Jaffe, W. C. Jones, E. Keihänen, R. Keskitalo, K. Kiiveri, J. Kim, T. S. Kisner, N. Krachmalnicoff, M. Kunz, H. Kurki-Suonio, G. Lagache, J.-M. Lamarre, A. Lasenby, M. Lattanzi, C. R. Lawrence, M. Le Jeune, J. Lesgourgues, F. Levrier, A. Lewis, M. Liguori, P. B. Lilje, V. Lindholm, M. Lpez-Caniego, P. M. Lubin, Y.-Z. Ma, J. F. Macías-Pérez, G. Maggio, D. Maino, N. Mandolesi, A. Mangilli, A. Marcos-Caballero, M. Maris, P. G. Martin, E. Martínez-González, S. Matarrese, N. Mauri, J. D. McEwen, P. D. Meerburg, P. R. Meinhold, A. Melchiorri, A. Mennella, M. Migliaccio, S. Mitra, M.-A. Miville-Deschênes, D. Molinari, A. Moneti, L. Montier, G. Morgante, A. Moss,

M. Münchmeyer, P. Natoli, H. U. NÃÿrgaard-Nielsen, L. Pagano, D. Paoletti, B. Partridge, G. Patanchon, H. V. Peiris, F. Perrotta, V. Pettorino, F. Piacentini, L. Polastri, G. Polenta, J.-L. Puget, J. P. Rachen, M. Reinecke, M. Remazeilles, A. Renzi, G. Rocha, C. Rosset, G. Roudier, J. A. Rubiño Martín, B. Ruiz-Granados, L. Salvati, M. Sandri, M. Savelainen, D. Scott, E. P. S. Shellard, M. Shiraishi, C. Sirignano, G. Sirri, L. D. Spencer, R. Sunyaev, A.-S. Suur-Uski, J. A. Tauber, D. Tavagnacco, M. Tenti, L. Toffolatti, M. Tomasi, T. Trombetti, J. Valiviita, B. Van Tent, P. Vielva, F. Villa, N. Vittorio, B. D. Wandelt, I. K. Wehus, S. D. M. White, A. Zacchei, J. P. Zibin, and A. Zonca. Planck 2018 results. X. Constraints on inflation. *ArXiv e-prints*, 1807:arXiv:1807.06211, July 2018. URL http://adsabs.harvard.edu/abs/2018arXiv180706211P. 3

Francisco Prada, Anatoly A. Klypin, Antonio J. Cuesta, Juan E. Betancort-Rijo, and Joel Primack. Halo concentrations in the standard LCDM cosmology, April 2011. URL http://adsabs.harvard.edu/abs/2011arXiv1104.5130P. 3, 12.6.9, 12.11.3, 12.11.3, 16.4.1, 18.1, 18.1

William H. Press and Paul Schechter. Formation of galaxies and clusters of galaxies by Self-Similar gravitational condensation. *Astrophysical Journal*, 187:425–438, February 1974. URL http://adsabs.harvard.edu/abs/1974ApJ...187..425P. 12.6.10, 12.6.11, 16.4.1, 18.1

M. L. Prevot, J. Lequeux, L. Prevot, E. Maurice, and B. Rocca-Volmerange. The typical interstellar extinction in the small magellanic cloud. *Astronomy and Astrophysics*, 132:389–392, March 1984. ISSN 0004-6361. URL http://adsabs.harvard.edu/abs/1984%26A...132..389P. 16.4.1, 18.1

E. Retana-Montenegro, E. van Hese, G. Gentile, M. Baes, and F. Frutos-Alfaro. Analytical properties of Einasto dark matter haloes. *Astronomy and Astrophysics*, 540:A70, April 2012. doi: 10.1051/0004-6361/201118543. URL https://ui.adsabs.harvard.edu/abs/2012%26A...540A..70R/abstract. 18.1

Luciano Rezzolla, Enrico Barausse, Ernst Nils Dorband, Denis Pollney, Christian Reisswig, Jennifer Seiler, and Sascha Husa. Final spin from the coalescence of two black holes. *Physical Review D*, 78:44002, August 2008. URL http://adsabs.harvard.edu/abs/2008PhRvD..78d4002R. 12.57.1, 16.4.1, 18.1

Massimo Ricotti and J. Michael Shull. Feedback from galaxy formation: Escaping ionizing radiation from galaxies at high redshift. *The Astrophysical Journal*, 542:548–558, October 2000. doi: 10.1086/317025;. URL http://adsabs.harvard.edu/abs/2000ApJ...542..548R. 12.21.2, 16.4.1, 18.1

Aldo Rodríguez-Puebla, Peter Behroozi, Joel Primack, Anatoly Klypin, Christoph Lee, and Doug Hellinger. Halo and subhalo demographics with Planck cosmological parameters: Bolshoi-Planck and MultiDark-Planck simulations. *Monthly Notices of the Royal Astronomical Society*, 462:893–916, October 2016. ISSN 0035-8711. doi: 10.1093/mnras/stw1705. URL http://adsabs.harvard.edu/abs/2016MNRAS.462..893R. 16.4.1

Elke Roediger and Marcus BrÃČÅŠggen. Ram pressure stripping of disc galaxies orbiting in clusters - i. mass and radius of the remaining gas disc. *Monthly Notices of the Royal Astronomical Society*, 380:1399–1408, October 2007. ISSN 0035-8711. doi: 10.1111/j.1365-2966.2007.12241.x;. URL http://adsabs.harvard.edu/abs/2007MNRAS.380.1399R. 12.25.2, 18.1

Laura V. Sales, Julio F. Navarro, Mario G. Abadi, and Matthias Steinmetz. Cosmic ménage à trois: the origin of satellite galaxies on extreme orbits. *Monthly Notices of the Royal Astronomical Society*, 379:1475–1483, August 2007. URL http://adsabs.harvard.edu/abs/2007MNRAS.379.1475S. 5.2.1

Edwin E. Salpeter. The luminosity function and stellar evolution. *The Astrophysical Journal*, 121:161, January 1955. URL http://adsabs.harvard.edu/abs/1955ApJ...121..161S. 12.28.2, 16.4.1, 18.1

J. M. Scalo. The stellar initial mass function. *Fundamentals of Cosmic Physics*, 11:1–278, May 1986. URL http://adsabs.harvard.edu/abs/1986FCPh...11....1S. 12.28.2, 16.4.1, 18.1

Maarten Schmidt. The rate of star formation. *The Astrophysical Journal*, 129:243, March 1959. URL http://adsabs.harvard.edu/abs/1959ApJ...129..243S. 12.45.1

Aurel Schneider. Structure formation with suppressed small-scale perturbations. *Monthly Notices of the Royal Astronomical Society*, 451:3117–3130, August 2015. ISSN 0035-8711. doi: 10.1093/mnras/stv1169. URL http://adsabs.harvard.edu/abs/2015MNRAS.451.3117S. 3, 12.11.3, 16.4.1, 18.1

Aurel Schneider, Robert E. Smith, Andrea V. Macciò, and Ben Moore. Non-linear evolution of cosmological structures in warm dark matter models. *Monthly Notices of the Royal Astronomical Society*, 424:684–698, July 2012. doi: 10.1111/j.1365-2966.2012.21252.x;. URL http://adsabs.harvard.edu/abs/2012MNRAS.424..684S. 12.11.3, 16.4.1, 18.1

T. T. Scholz and H. R. J. Walters. Collisional rates and cooling within atomic hydrogen plasmas. *The Astrophysical Journal*, 380:302–306, October 1991. ISSN 0004-637X. doi: 10.1086/170587. URL http://adsabs.harvard.edu/abs/1991ApJ...380..302S. 16.4.1, 18.1

Sara Seager, Dimitar D. Sasselov, and Douglas Scott. How exactly did the universe become neutral? *The Astrophysical Journal Supplement Series*, 128:407–430, June 2000. URL http://adsabs.harvard.edu/abs/2000ApJS..128..407S. 12.29.2

J. L. Sérsic. Influence of the atmospheric and instrumental dispersion on the brightness distribution in a galaxy. *Boletin de la Asociacion Argentina de Astronomia La Plata Argentina*, 6:41, 1963. URL http://adsabs.harvard.edu/abs/1963BAAA....6...41S. 11.5.2

N. I. Shakura and R. A. Sunyaev. Black holes in binary systems. observational appearance. *Astronomy and Astrophysics*, 24:337–355, 1973. URL http://ukads.nottingham.ac.uk/abs/1973%26A....24..337S. 12.5.1, 12.5.3, 16.4.1, 18.1

Paul R. Shapiro and Hyesung Kang. Hydrogen molecules and the radiative cooling of pregalactic shocks. *The Astrophysical Journal*, 318:32–65, July 1987. URL http://adsabs.harvard.edu/abs/1987ApJ...318...32S. 18.1, *a*

Shiyin Shen, H. J. Mo, Simon D. M. White, Michael R. Blanton, Guinevere Kauffmann, Wolfgang Voges, J. Brinkmann, and Istvan Csabai. The size distribution of galaxies in the sloan digital sky survey. *Monthly Notices of the Royal Astronomical Society*, 343:978–994, August 2003. URL http://adsabs.harvard.edu/abs/2003MNRAS.343..978S. 9.5.3, 9.5.3, 9.5.3, 9.5.3

Ravi K. Sheth, H. J. Mo, and Giuseppe Tormen. Ellipsoidal collapse and an improved model for the number and spatial distribution of dark matter haloes. *Monthly Notices of the Royal Astronomical Society*, 323:1–12, May 2001. URL http://adsabs.harvard.edu/abs/2001MNRAS.323....1S. 3, 12.6.10, 12.6.11, 12.6.11, 12.15.3, 15.8.2, 15.8.2, 16.4.1, 16.4.1, 16.4.1, 18.1, 18.1

Yong Shi, George Helou, Lin Yan, Lee Armus, Yanling Wu, Casey Papovich, and Sabrina Stierwalt. Extended schmidt law: Role of existing stars in current star formation. *The Astrophysical Journal*, 733:87, June 2011. URL http://adsabs.harvard.edu/abs/2011ApJ...733...87S. 3, 12.45.2, 16.4.1, 18.1

J. M. Shull and M. van Steenberg. The ionization equilibrium of astrophysically abundant elements. *The Astrophysical Journal Supplement Series*, 48:95–107, January 1982. URL http://adsabs.harvard.edu/abs/1982ApJS...48...95S. 16.4.1, 18.1

R. E. Smith and P. I. R. Watts. Triaxial haloes, intrinsic alignments and the dark matter power spectrum. *Monthly Notices of the Royal Astronomical Society*, 360:203–215, June 2005. ISSN 0035-8711. doi: 10.1111/j.1365-2966.2005.09053.x. URL http://adsabs.harvard.edu/abs/2005MNRAS.360..203S. 12.20.2, 16.4.1, 18.1

Robert E. Smith. How covariant is the galaxy luminosity function?, May 2012. URL http://adsabs.harvard.edu/abs/2012arXiv1205.4240S. 9.5.3, 3

David Sobral, Ian Smail, Philip N. Best, James E. Geach, Yuichi Matsuda, John P. Stott, Michele Cirasuolo, and Jaron Kurk. A large HÎś survey at z = 2.23, 1.47, 0.84 and 0.40: the 11 Gyr evolution of star-forming galaxies from HiZELS. *Monthly Notices of the Royal Astronomical Society*, 428:1128–1146, January 2013. ISSN 0035-8711. doi: 10.1093/mnras/sts096. URL http://adsabs.harvard.edu/abs/2013MNRAS.428.1128S. 16.4.1

C. Srisawat, A. Knebe, F. R. Pearce, A. Schneider, P. A. Thomas, P. Behroozi, K. Dolag, P. J. Elahi, J. Han, J. Helly, Y. Jing, I. Jung, J. Lee, Y.-Y. Mao, J. Onions, V. Rodriguez-Gomez, D. Tweed, and S. K. Yi. Sussing merger trees: The merger trees comparison project. *Monthly Notices of the Royal Astronomical Society*, October 2013. ISSN 0035-8711, 1365-2966. doi: 10.1093/mnras/stt1545. URL http://adsabs.harvard.edu/doi/10.1093/mnras/stt1545. 3, 12.37.2, 16.4.1, 18.1

A. A. Suchkov and Yu. A. Shchekinov. Cooling of hot protogalactic gas by the h2/+/ molecular ion. *Soviet Astronomy Letters*, 4:164, April 1978. URL http://adsabs.harvard.edu/abs/1978SvAL....4..164S. 12.9.1, 18.1

Ralph S. Sutherland. Accurate free-free gaunt factors for astrophysical plasmas. *Monthly Notices of the Royal Astronomical Society*, 300:321–330, October 1998. ISSN 0035-8711. doi: 10.1046/j.1365-8711.1998.01687.x. URL http://adsabs.harvard.edu/abs/1998MNRAS.300..321S. 16.4.1, 18.1

Ryuichi Takahashi, Masamune Oguri, Masanori Sato, and Takashi Hamana. Probability distribution functions of cosmological lensing: Convergence, shear, and magnification. *The Astrophysical Journal*, 742:15, November 2011. doi: 10.1088/0004-637X/742/1/15;. URL http://adsabs.harvard.edu/abs/2011ApJ...742...15T. 12.19.1, 16.4.1, 18.1

H. Takeda, P. E. J. Nulsen, and A. C. Fabian. Ram pressure stripping in a changing environment. *Monthly Notices of the Royal Astronomical Society*, 208:261–278, May 1984. ISSN 0035-8711. URL http://adsabs.harvard.edu/abs/1984MNRAS.208..261T. 12.40.1, 18.1

Max Tegmark, Joseph Silk, Martin J. Rees, Alain Blanchard, Tom Abel, and Francesco Palla. How small were the first cosmological objects? *ApJ*, 474:1, 1997. URL http://adsabs.harvard.edu/abs/1997ApJ...474....1T. 12.39.2, 16.4.1, 18.1

Cajo J. F. Terr Braak. A markov chain monte carlo version of the genetic algorithm differential evolution: easy bayesian computing for real parameter spaces. *Stat Comput*, 16(3):239–249, September 2006. ISSN 0960-3174, 1573-1375. doi: 10.1007/s11222-006-8769-1. URL http://link.springer.com/article/10.1007/s11222-006-8769-1. 9.7.9

J. Thorley, M. Wilkinson, and M. Charleston. The information content of consensus trees. In *Studies in Classification, Data Analysis, and Knowledge Organization*, Advances in Data Science and Classification, pages 91–98, Berlin, 1998. Springer-Verlag. 16.4.1, 18.1

Jeremy Tinker, Andrey V. Kravtsov, Anatoly Klypin, Kevork Abazajian, Michael Warren, Gustavo Yepes, Stefan Gottlöber, and Daniel E. Holz. Toward a halo mass function for precision cosmology: The limits of universality. *The Astrophysical Journal*, 688:709–728, December 2008. URL http://adsabs.harvard.edu/abs/2008ApJ...688..709T. 3, 12.6.11, 15.8.2, 15.8.2, 15.8.2, 15.8.2, 16.4.1, 18.1

Jeremy L. Tinker, Brant E. Robertson, Andrey V. Kravtsov, Anatoly Klypin, Michael S. Warren, Gustavo Yepes, and Stefan Gottlober. The large scale bias of dark matter halos: Numerical calibration and model tests. http://adsabs.harvard.edu/abs/2010arXiv1001.3162T, January 2010. URL http://adsabs.harvard.edu/abs/2010arXiv1001.3162T. 12.6.10, 16.4.1, 18.1

Adam R. Tomczak, Ryan F. Quadri, Kim-Vy H. Tran, Ivo Labbé, Caroline M. S. Straatman, Casey Papovich, Karl Glazebrook, Rebecca Allen, Gabriel B. Brammer, Glenn G. Kacprzak, Lalitwadee Kawinwanichakij, Daniel D. Kelson, Patrick J. McCarthy, Nicola Mehrtens, Andrew J. Monson, S. Eric Persson, Lee R. Spitler, Vithal Tilvi, and Pieter van Dokkum. Galaxy stellar mass functions from ZFOURGE/CANDELS: an excess of low-mass galaxies since z = 2 and the rapid buildup of quiescent galaxies. *The Astrophysical Journal*, 783:85, March 2014. ISSN 0004-637X. doi: 10.1088/0004-637X/783/2/85. URL http://adsabs.harvard.edu/abs/2014ApJ...783...85T. 3, 9.5.3, 12.4, 12.59.4, 16.4.1, 18.1

Giuseppe Tormen. The rise and fall of satellites in galaxy clusters. *Monthly Notices of the Royal Astronomical Society*, 290:411–421, September 1997. URL http://adsabs.harvard.edu/abs/1997MNRAS.290..411T. 12.51.1, 16.4.1, 18.1

Hy Trac, Renyue Cen, and Philip Mansfield. SCORCH I: The Galaxy-Halo Connection in the First Billion Years. *The Astrophysical Journal*, 813:54, November 2015. ISSN 0004-637X. doi: 10.1088/0004-637X/813/1/54. URL http://adsabs.harvard.edu/abs/2015ApJ...813...54T. 3

Michele Trenti, Britton D. Smith, Eric J. Hallman, Samuel W. Skillman, and J. Michael Shull. How Well do Cosmological Simulations Reproduce Individual Halo Properties? *The Astrophysical Journal*, 711:1198–1207, March 2010. ISSN 0004-637X. doi: 10.1088/0004-637X/711/2/1198. URL http://adsabs.harvard.edu/abs/2010ApJ...711.1198T. 16.4.1, 18.1

Frank C. van den Bosch, Giuseppe Tormen, and Carlo Giocoli. The mass function and average mass-loss rate of dark matter subhaloes. *Monthly Notices of the Royal Astronomical Society*, 359:1029–1040, May 2005. URL http://adsabs.harvard.edu/abs/2005MNRAS.359.1029V. 3, 12.11.5, 12.11.5, 16.4.1, 18.1

P. A. M. van Hoof, R. J. R. Williams, K. Volk, M. Chatzikos, G. J. Ferland, M. Lykins, R. L. Porter, and Y. Wang. Accurate determination of the free-free Gaunt factor - I. Non-relativistic Gaunt factors. *Monthly Notices of the Royal Astronomical Society*, 444:420–428, October 2014. ISSN 0035-8711. doi: 10.1093/mnras/stu1438. URL http://adsabs.harvard.edu/abs/2014MNRAS.444..420V. 16.4.1, 18.1

D. A. Verner and G. J. Ferland. Atomic data for astrophysics. i. radiative recombination rates for h-like, he-like, li-like, and na-like ions over a broad range of temperature. *The Astrophysical Journal Supplement Series*, 103:467, April 1996. URL http://adsabs.harvard.edu/abs/1996ApJS..103..467V. 18.1

D. A. Verner and D. G. Yakovlev. Analytic FITS for partial photoionization cross sections. *Astronomy and Astrophysics Supplement Series*, 109:125–133, January 1995. URL http://adsabs.harvard.edu/abs/1995%26AS..109..125V. 18.1

D. A. Verner, G. J. Ferland, K. T. Korista, and D. G. Yakovlev. Atomic data for astrophysics. II. new analytic FITS for photoionization cross sections of atoms and ions. *The Astrophysical Journal*, 465:487, July 1996. URL http://adsabs.harvard.edu/abs/1996ApJ...465..487V. 18.1

Á. Villalobos, G. De Lucia, S. M. Weinmann, S. Borgani, and G. Murante. An improved prescription for merger time-scales from controlled simulations. *Monthly Notices of the Royal Astronomical Society*, 433:L49–L53, June 2013. ISSN 0035-8711. doi: 10.1093/mnrasl/slt056;. URL http://adsabs.harvard.edu/abs/2013MNRAS.433L..49V. 12.51.1, 16.4.1, 18.1

Maya Vitvitska, Anatoly A. Klypin, Andrey V. Kravtsov, Risa H. Wechsler, Joel R. Primack, and James S. Bullock. The Origin of Angular Momentum in Dark Matter Halos. *Astrophysical Journal*, 581:799–809, December 2002. URL http://adsabs.harvard.edu/abs/2002ApJ...581..799V. 11.12.4, 18.1

Marta Volonteri, Francesco Haardt, and Piero Madau. The assembly and merging history of supermassive black holes in hierarchical models of galaxy formation. *The Astrophysical Journal*, 582:559–573, January 2003. URL http://adsabs.harvard.edu/abs/2003ApJ...582..559V. 11.1.1, 12.54.2, 12.55.2, 12.55.2, 16.4.1, 16.4.1, 18.1, 18.1

G. S. Voronov. A practical fit formula for ionization rate coefficients of atoms and ions by electron impact: Z = 1-28. *Atomic Data and Nuclear Data Tables*, 65:1, 1997. URL http://adsabs.harvard.edu/abs/1997ADNDT..65....1V. 18.1

Yogesh Wadadekar, Braxton Robbason, and Ajit Kembhavi. Two-dimensional galaxy image decomposition. *The Astronomical Journal*, 117:1219–1228, March 1999. URL http://adsabs.harvard.edu/abs/1999AJ....117.1219W. 11.5.2

Risa H. Wechsler, James S. Bullock, Joel R. Primack, Andrey V. Kravtsov, and Avishai Dekel. Concentrations of dark halos from their assembly histories. *The Astrophysical Journal*, 568:52–70, March 2002. URL http://adsabs.harvard.edu/abs/2002ApJ...568...52W. 3, 12.11.1, 12.11.1, 16.4.1, 18.1, 18.1

Nevin N. Weinberg and Marc Kamionkowski. Constraining dark energy from the abundance of weak gravitational lenses. *Monthly Notices of the Royal Astronomical Society*, 341:251–262, May 2003. ISSN 0035-8711. doi: 10.1046/j.1365-8711.2003.06421.x;. URL http://adsabs.harvard.edu/abs/2003MNRAS.341..251W. 18.1

Andrew R. Wetzel. On the orbits of infalling satellite halos. http://adsabs.harvard.edu/abs/2010arXiv1001.4792W, January 2010. URL http://adsabs.harvard.edu/abs/2010arXiv1001.4792W. 12.51.2, 16.4.1, 18.1

Andrew R. Wetzel and Martin White. What determines satellite galaxy disruption? *Monthly Notices of the Royal Astronomical Society*, 403:1072–1088, April 2010. URL http://adsabs.harvard.edu/abs/2010MNRAS.403.1072W. 12.51.1, 16.4.1, 18.1

Simon D. M. White and Carlos S. Frenk. Galaxy formation through hierarchical clustering. *Astrophysical Journal*, 379:52–79, September 1991. URL http://adsabs.harvard.edu/abs/1991ApJ...379...52W. 3, 12.9.2, 12.9.9, 16.4.1, 16.4.1, 18.1

Michael Wichura. The percentage points of the normal distribution. *Applied Statistics*, 37(3):477–484, 1988. 18.1

J. Wilms, A. Allen, and R. McCray. On the absorption of x-rays in the interstellar medium. *The Astrophysical Journal*, 542:914–924, October 2000. doi: DOI:10.1086/317016;eprintid:arXiv:astro-ph/0008425. URL http://adsabs.harvard.edu/abs/2000ApJ...542..914W. 18.1

Adolf N. Witt and Karl D. Gordon. Multiple scattering in clumpy media. II. galactic environments. *The Astrophysical Journal*, 528:799–816, January 2000. ISSN 0004-637X. doi: 10.1086/308197. URL http://adsabs.harvard.edu/abs/2000ApJ...528..799W. 3, 16.4.1, 18.1

Wan Yan Wong, Adam Moss, and Douglas Scott. How well do we understand cosmological recombination? *Monthly Notices of the Royal Astronomical Society*, 386:1023–1028, May 2008. URL http://adsabs.harvard.edu/abs/2008MNRAS.386.1023W. 12.29.2

Andrew R. Zentner, Andreas A. Berlind, James S. Bullock, Andrey V. Kravtsov, and Risa H. Wechsler. The physics of galaxy clustering. i. a model for subhalo populations. *The Astrophysical Journal*, 624: 505–525, May 2005. ISSN 0004-637X. doi: 10.1086/428898;. URL http://adsabs.harvard.edu/abs/2005ApJ...624..505Z. 12.43.2, 16.4.1, 18.1

Jun Zhang and Lam Hui. On random walks with a general moving barrier. *The Astrophysical Journal*, 641:641–646, April 2006. URL http://adsabs.harvard.edu/abs/2006ApJ...641..641Z. 12.14.3, 12.14.4, 15.8.2, 15.8.2, 16.4.1, 18.1

D. H. Zhao, Y. P. Jing, H. J. Mo, and G. Börner. Accurate universal models for the mass accretion histories and concentrations of dark matter halos. *The Astrophysical Journal*, 707:354–369, December 2009. URL http://adsabs.harvard.edu/abs/2009ApJ...707..354Z. 12.11.1, 12.11.3, 12.11.3, 16.4.1, 16.4.1, 18.1, 18.1, 18.1

M. A. Zwaan, M. J. Meyer, L. Staveley-Smith, and R. L. Webster. The HIPASS catalogue: $\omega_{\rm HI}$ and environmental effects on the HI mass function of galaxies. *Monthly Notices of the Royal Astronomical Society*, 359:L30–L34, May 2005. URL http://adsabs.harvard.edu/abs/2005MNRAS.359L..30Z. 4.4.1

Martin A. Zwaan, Frank H. Briggs, David Sprayberry, and Ertu Sorar. The h i mass function of galaxies from a deep survey in the 21 centimeter line. *The Astrophysical Journal*, 490:173, November 1997. ISSN 0004-637X. doi: 10.1086/304872. URL http://adsabs.harvard.edu/abs/1997ApJ...490..173Z. 4.4.1

Glossary

MANGLE MANGLE is a software package used for defining angular masks on the surface of a sphere. It is used primarily for defining the geometry of galaxy surveys. 249, 399, 403, 404, 1697–1699, 1705, 1706, 1709–1712, 1715, 1717, 1719, 1720, 1724, 1725, 1727, 1730, 2178, 2579–2583, 2585, 2590, 2592, 2595–2598, 2601, 2602

AB magnitude An astronomical magnitude system in which the apparent magnitude is defined as $m = -2.5 \log_{10} f - 48.60$ for a flux density, f, measured in ergs per second per square centimeter per hertz. 282

ADAF An advection-dominated accretion flow (ADAF) is a particular solution for an accretion flow around a black hole, star or compact object in which energy liberated by viscous forces is stored within the accretion flow and advected inward to the central object (see Narayan et al. 1998). 3909

AHF Amiga's Halo Finder (AHF) is a software package which identifies dark matter halos in N-body simulations. Full details are given by Gill et al. [2004] and Knollmann and Knebe [2009]. 3909

Approximate Nearest Neighbor The Approximate Nearest Neighbor library is a software package used finding the closest set of points to a given point, approximately. 267

backward descendent The primary progenitor of a node. This type of descendent is usually relevant when building merger trees and should be distinguished from a forward descendent which is relevant when considering how halos and galaxies evolve forward in time. 3906

BIE The Bayesian Inference Engine (BIE) is a software package designed to facilitate exploration of complexes parameter spaces using Bayesian techniques.. 3909

CAMB CAMB is a code to compute anisotropies in the cosmic microwave background, and the linear theory power spectra of matter and radiation. 2633

CDF Cumulative Distribution Function (CDF) is a function which describes the cumulative probability for a random variable to be equal to or less than a given value.. 3909

CDM Cold dark matter (CDM) is a hypothesized type of dark matter in which the particles move slowly compared to the speed of light. 3909

Cloudy Cloudy is a code to compute the ionization structure of HII regions. 2634

component An individual physical system within a node, such as a dark matter halo, a galactic disk or a supermassive black hole. 286, 289, 291–293, 295, 297, 299, 301, 303, 304, 308–312, 315, 317–321, 356–361, 387, 398, 3906

deadlock A deadlock describes a situation in which no node in a merger tree (or forest) can be evolved further forward in time due to the existence of circular dependencies between nodes. Deadlocks can occur due to incompatible parameter choices, or may indicate a bug in Galacticus.. 216

DSL Domain-specific languages (DSL) are a type of programming language dedicated to a particular problem. In Galacticus a DSL is used to specify the structure of components. 3909

ENZO Enzo is an adaptive mesh refinement hydrodynamics code. 363, 366

FFTLog FFTLog is a code to compute fast Fourier transforms of discrete periodic sequences of logarithmically spaced data. 2874

forest A collection of merger trees that are linked together by virtue of nodes which jump between trees. 211, 216

forward descendent The node with which the mass (or majority of the mass) of a node will become associated with at a later time. This type of descendent is usually relevant when considering how halos and galaxies evolve forward in time and should be distinguished from a backward descendent which is relevant when building merger trees. 409, 3905

GAMA The Galaxy and Mass Assembly (GAMA) survey is a spectroscopic survey of $\approx 300,000$ galaxies down to $r < 19.8$ mag over ≈ 286 deg^2.. 3909

GraphViz GraphViz is an open source graph visualization package. It is used by Galacticus in making diagrams of merger trees.. 424, 425, 428, 2032, 2760, 2761

GSL GNU Scientific Library (GSL) is a library providing a variety of numerical algorithms.. 3909

HDF5 The hierarhical data format (version 5; HDF5) is a file format designed for storing scientific data.. 3909

HOD A halo occupation distribution (HOD) is a mathematical model describing the distribution of the number of galaxies (of some given physical properties) found in a dark matter halo of given mass.. 3909

IRATE IRATE is file format designed for N-body simulation particle, halo, merger tree, and galaxy data. 790, 3765, 3766

Latin hypercube A Latin hypercube is a construct for generating a sample of plausible collections of parameter values from a multidimensional distribution. 270, 3906

maximin In Latin hypercube design, the "maximin" approach generates a large number of Latin hypercubes and selects the one which has the greatest minimum distance between all pairs of points in the hypercube. 270

MD5 hash The MD5 Message-Digest Algorithm is a widely used cryptographic hash function that produces a 128-bit (16-byte) hash value. In Galacticus it is used to encode unique labels for modules which are incorporated into file names. Galacticus uses the `glibc crypt()` function to compute MD5 hashes, but switches "`/`" for "`@`" in the hash (since "`/`" is inconvenient for use in file names). 431

mergee For a given node in a merger tree, the set of mergee nodes consists of all nodes which will undergo a galaxy merger with the node at some point in the future. 232

Millennium Simulation The Millennium Simulation is a high-resolution N-body simulation of structure formation in a cold dark matter universe.. 239

MPI Message Passing Interface (MPI) is a standard for passing messages between processes on parallel computers.. 3909

node A single point in a merger tree, consisting of a dark matter halo and associated baryons. 210, 216, 217, 231, 232, 287, 288, 291, 292, 294, 295, 304–308, 313, 314, 317, 318, 320, 342, 343, 347, 356, 357, 374, 381, 394, 438, 567–570, 908–913, 1886, 2043, 2340, 2341, 3122, 3343, 3723, 3733, 3734, 3905, 3907

OpenMP OpenMP is an API for shared memory parallel programming.. 445

PAH Polycyclic aromatic hydrocarbons (PAH) are large organic molecules consisting of fused aromatic rings. 3909

parent In a merger tree, the parent node of any given node that exists at time t_0 is that node to which it is directly connected in the tree at time $t_1 > t_0$. 231, 232

PBS Portable Batch System (PBS) is a job scheduler used on many compute cluster environments.. 3909

PDF Probability Density Function (PDF) is a function which describes the probability for a random variable to take on a given value.. 3909

primary progenitor The progenitor of a given node which is regarding as the direct descendent of that node (often, but not always, the most massive progenitor). Other progenitors are considered to merge into this primary progenitor. 231, 232, 3905

RecFast RecFast is a code to calculate the recombination history of the universe. 2637

SAM Semi-analytic models (SAMs) are a type of galaxy formation model utilizing simple parameterizations of physical processes to follow the evolution of galaxies through a merging hierarchy of galaxies.. 3910

WDM Warm dark matter (WDM) is a hypothesized type of dark matter in which the particle has non-negligible thermal velocity at decoupling. 3910

Acronyms

ADAF advection-dominated accretion flow. 326, 327, *Glossary:* ADAF

AHF Amiga's Halo Finder. 379, *Glossary:* AHF

BIE semi-analytic model. *Glossary:* BIE

CDF cumulative distribution function. 715–717, 719, 1950, *Glossary:* CDF

CDM cold dark matter. 3, 19, 41, 330, 377, 2698, 3671–3673, 3675, 3677–3679, *Glossary:* CDM

CMB cosmic microwave background. 155, 336, 1854, 2288, 2292, 2293, 2334

DSL domain-specific language. *Glossary:* DSL

GAMA Galaxy and Mass Assembly. 130, 131, *Glossary:* GAMA

GSL GNU Scientific Library. 2875, 2876, 2900, 2901, 2904, 2905, *Glossary:* GSL

HDF5 hierarchical data format. 2030, *Glossary:* HDF5

HOD halo occupation distribution. 249, 250, 257, 258, *Glossary:* HOD

IGM intergalactic medium. 29, 59, 120, 134, 226–228, 293, 323, 324, 388, 389, 469–477, 479, 480, 783–790, 1815–1818, 1980–1983, 2235, 2236, 2238, 2240, 2242, 2243, 2639–2646, 3707

IMF initial mass function. 93, 97, 100, 101, 142, 225, 366–368, 387, 388, 767–775, 1977, 2635, 3581–3586

ISCO innermost stable circular orbit. 326

ISM interstellar medium. 206, 286, 390, 2233, 2234, 3606

MCMC Markov Chain Monte Carlo. 249, 260, 264, 274

MPI message passing interface. 260, 272, *Glossary:* MPI

NFW Navarro-Frenk-White (dark matter halo profile). 51, 333, 342, 343, 346, 363

ODE ordinary differential equation. 231, 323, 1665–1667, 1805, 2153

PAH polycyclic aromatic hydrocarbon. *Glossary:* PAH

PBS portable batch system. 260, *Glossary:* PBS

PDF probability density function. 716–719, 1949, 3635, 3638, 3640, *Glossary:* PDF

SAM semi-analytic model. 252, 399, 406, *Glossary:* SAM

SDSS Sloan Digitial Sky Survey. 250, 257, 259

SNe supernovae. 171, 225

WDM warm dark matter. 1885, 2187, 2188, 2381, 3618, 3672, 3673, *Glossary:* WDM

Index

Code Index